7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 - Paris - France
«Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination»

EQUILIBRAGE DES LIGNES D'USINAGE A BOITIERS MULTI-BROCHES AVEC LA METHODE GRASP

Olga GUSCHINSKAYA, Alexandre DOLGUI

Ecole des Mines de Saint-Etienne
158, Cours Fauriel
42100 Saint-Etienne
guschinskaya@yahoo.fr, dolgui@emse.fr

RÉSUMÉ : *Ce travail porte sur l'équilibrage des lignes d'usinage lors de leur conception. Une telle ligne est constituée des postes de travail avec des boîtiers multibroches qui exécutent plusieurs opérations en parallèle. Le problème d'optimisation consiste à déterminer le nombre minimal d'équipements (boîtiers multibroches et postes de travail) nécessaires pour assurer la fabrication de la pièce tout en respectant les contraintes technologiques et économiques connues. Dans cet article, nous proposons une méthode de type GRASP pour la résolution de ce problème. Les performances de notre méthode ont été évaluées par des expérimentations numériques dont les résultats sont présentés et analysés.*

MOTS-CLÉS : *GRASP, Métaheuristique, Équilibrage, Optimisation, Conception, Lignes d'usinage*

1. INTRODUCTION

Les industries mécaniques se trouvent aujourd'hui confrontées à de nouveaux défis et dans un marché extrêmement compétitif. Afin de se donner les moyens de produire au plus juste coût, en maîtrisant qualité et délais, il est impératif d'effectuer de manière rationnelle les choix stratégiques. Dans cet article, nous étudions la problématique de conception de lignes de transfert. Plus précisément, nous nous intéressons au problème d'équilibrage d'une ligne d'usinage lors de la mise en fabrication d'un nouveau produit.

Les lignes d'usinage que nous considérons sont constituées d'une chaîne de postes de travail reliés par un dispositif de transfert de pièce. Ainsi, chaque pièce chargée sur la ligne passe par tous les postes de travail dans l'ordre de leur disposition ; sur chaque poste de travail, elle subit des opérations d'usinage. À intervalle de temps régulier, égal au temps de cycle, toutes les pièces se trouvant sur la ligne sont simultanément déplacées vers le poste de travail suivant. Au même moment, un nouveau brut est chargé sur le premier poste de travail et une pièce finie est déchargée sur le dernier poste. Afin d'assurer le déplacement synchrone de pièces et éviter l'utilisation de stocks tampons entre les postes, le temps d'usinage sur chaque poste de travail doit être inférieur au temps de cycle objectif, désigné par T_0.

Chaque poste de travail est muni d'au moins un boîtier. Les boîtiers peuvent comporter un ou plusieurs outils. Dans le dernier cas, il s'agit des boîtiers multibroches qui sont utilisés pour effectuer simultanément un ensemble d'opérations. Les boîtiers de chaque poste sont activés en séquence. Le changement du boîtier actif est effectué pendant un temps τ_b, le cas échéant, pendant ce temps la pièce peut être également repositionnée. Le changement de boîtiers actifs n'est pas synchronisé entre les différents postes de travail. L'ordre de déclenchement de boîtiers et l'affectation d'outils à chaque boîtier sont à définir lors de la conception de la ligne.

Le problème d'équilibrage de telles lignes consiste à répartir les opérations d'usinage nécessaires pour la fabrication de la pièce à des boîtiers d'usinage et à des postes de travail de sorte que toutes les contraintes techniques et technologiques soient respectées. Le nombre de postes à équiper et le nombre de boîtiers à installer définissent principalement le coût de la ligne que l'on vise à minimiser lors de la répartition des opérations.

L'article est organisé comme suit. Dans la section 2, nous présentons le problème d'optimisation de manière formelle et les approches déjà proposées pour sa résolution. La méthode GRASP est développée dans la section 3 et évaluée dans la section 4. Les conclusions et perspectives font l'objet de la section 5.

2. PROBLÈME D'OPTIMISATION

Nous supposons que pour chaque problème P les données suivantes sont connues :

N est l'ensemble des opérations d'usinage nécessaires pour la fabrication de la pièce;

$l_i, i \in \mathbf{N}$ est la longueur de la course d'outil, nécessaire pour l'opération i ;

$[v_{f1}(i), v_{f2}(i)]$ est l'intervalle des valeurs admissibles de la vitesse d'avance d'outil pour l'opération i ;

$v_{f0}(i)$ est la valeur « conseillée » de la vitesse d'avance d'outil, $v_{f0}(i) \in [v_{f1}(i), v_{f2}(i)]$;

T_0 est le temps de cycle d'usinage à ne pas dépasser (calculé sur la base de la productivité requise et du rendement de la ligne d'usinage) ;

τ^p est le temps auxiliaire au niveau d'un poste de travail ;

τ^b le temps auxiliaire au niveau d'un bloc ;

C_1 et C_2 sont les coûts relatifs d'un poste de travail et d'un bloc, respectivement;

m_0 est le nombre maximum autorisé de postes de travail du système;

n_0 le nombre total maximum autorisé de boîtiers par poste de travail ;

Les contraintes de précédence qui expriment l'ordre du déroulement des opérations, imposé par la technologie d'usinage utilisée. Ce type de contraintes peut être modélisé par un graphe orienté G^{pr}=($\mathbf{N}, D^{pr}$), dans lequel un arc $\{i, j\} \in D^{pr}$ si l'opération i ne peut pas être exécutée après l'exécution de l'opération j. Les contraintes de précédence sont « non strictes », cela signifie que si un arc $\{i, j\} \in G^{pr}$, alors les opérations i et j peuvent être affectées au même bloc, c'est-à-dire peuvent être exécutées **simultanément** par le même boîtier.

Les contraintes d'inclusion qui traduisent la nécessité d'exécuter certaines opérations sur le même poste de travail, notamment afin de respecter la tolérance de position requise. Ce type de contraintes peut être représenté par une collection I^p de sous-ensembles $e \subset \mathbf{N}$ d'opérations, tels que toutes les opérations appartenant à l'ensemble e doivent être impérativement affectées au même poste de travail.

Les contraintes d'exclusion au niveau des postes de travail qui viennent de l'impossibilité d'exécuter certaines opérations sur le même poste de travail. Ce type de contraintes peut être représenté par une collection E^p de sous-ensembles $e \subset \mathbf{N}$ d'opérations, tels qu'au moins une opération de chaque ensemble e ne doit pas être affectée au même poste de travail que les autres opérations de cet ensemble.

Les contraintes d'exclusion au niveau des blocs qui viennent de l'impossibilité d'exécuter certaines opérations par le même boîtier. Par exemple, les opérations correspondantes à deux faces différentes de la pièce ne peuvent pas être réalisées par le même boîtier. Ce type de contraintes peut être représenté par une collection E^b de sous-ensembles $e \subset \mathbf{N}$ d'opérations, tels qu'au moins une opération de chaque ensemble e ne doit pas être affectée au même bloc que les autres.

Nous introduisons les notations suivantes afin de pouvoir proposer une modélisation du problème d'optimisation que nous étudions : S une solution du problème générique ; m le nombre de postes de travail de la solution S ; k l'indice pour les postes de travail, $k = 1, 2, ..., m$; N_k l'ensemble des opérations affectées au poste de travail avec l'indice k ; b_k le nombre de blocs du poste de travail k ; b l'indice pour les blocs d'un poste de travail, pour le poste k : $b = 1, 2, ..., b_k$; N_{kb} l'ensemble des opérations affectées au bloc b du poste de travail k. Avec ces notations, une solution admissible du problème P peut être représentée par une collection $S(P) = \{\{N_{11}, ..., N_{1b_1}\}, ..., \{N_{m1}, ..., N_{mb_m}\}\}$.

Soit $\mathbf{S}(P)$ l'ensemble de toutes les solutions admissibles pour un problème P, c'est-à-dire toutes les solutions respectant l'ensemble des contraintes du problème P. La fonction objectif $C(S)$ prend la forme représentée par (1). La collection $S_{opt}(P) \in \mathbf{S}(P)$ telle que $C(S_{opt}) \leq C(S)\ \forall S(P) \in \mathbf{S}(P)$ correspond à la solution optimale du problème P.

Le problème d'optimisation étudié dans cet article peut être formulé de la manière suivante :

- Fonction objectif :

$$Minimiser\ C(S) = C_1 m + C_2 \sum_{k=1}^{m} b_k \tag{1}$$

S. c. :

Contraintes sur l'affectation de chaque opération exactement à un bloc :

$$\bigcup_{k=1}^{m} \bigcup_{b=1}^{b_k} N_{kb} = \mathbf{N} \tag{2}$$

$$N_{k'b'} \cap N_{k''b''} = \emptyset, \forall k', k'' = 1, ..., m, \\ b' = 1, ..., b_{k'}, b'' = 1, ..., b_{k''}, \\ \text{où } (k'b') \neq (k''b'') \tag{3}$$

- Contrainte sur le temps de cycle :

$$T(S) \leq T_0 \tag{4}$$

- Contraintes de précédence :

$$(k'-1)n_0 + b' \leq (k''-1)n_0 + b'', \quad i \in N_{k'b'}, j \in N_{b''k''}, \forall\, i \in PredD(j) \tag{5}$$

- Contraintes d'inclusion :

$$N_k \cap e \in \{\emptyset, e\}, \text{pour } \forall e \in I^p, k = 1, ..., m \tag{6}$$

- Contraintes d'exclusion au niveau des postes de travail :

$$e \not\subset N_b, \text{pour } \forall e \in E^p, k = 1, ..., m \tag{7}$$

- Contraintes d'exclusion au niveau des blocs :

$$e \not\subset N_{pb}, \text{pour } \forall e \in E^b, k = 1, ..., m, b = 1, ..., b_k \tag{8}$$

- Contrainte sur le nombre de postes de travail :

$$m \leq m_0 \tag{9}$$

- Contraintes sur le nombre de blocs par poste de travail:

$$b_k \leq n_0, \text{pour } k = 1, ..., m \tag{10}$$

Ce problème d'optimisation a été introduit dans (Dolgui et al., 1999). Dans ce qui suit, nous présentons un aperçu des travaux qui ont déjà été entrepris afin de développer des méthodes efficaces pour la résolution de ce type de problème d'optimisation.

Dans un premier temps, l'intérêt a été porté sur le développement de méthodes exactes. La première approche qui a été développée consistait en la transformation du problème initial en un problème de recherche du plus court chemin avec contraintes dans un graphe spécialement construit (Dolgui et al., 1999). Dans cette approche, la difficulté principale réside dans la construction de ce graphe (Dolgui et al., 2008). Ensuite, le problème a été présenté sous forme d'un programme linéaire en variables mixtes (MIP) en vue de le résoudre à l'aide du logiciel d'optimisation Cplex d'ILOG (Finel, 2004).

L'étude des performances de ces deux méthodes exactes a montré qu'elles sont complémentaires (Guschinskaya et Dolgui, 2006). Face à des problèmes ayant beaucoup de contraintes, la taille du graphe de décisions, utilisé par la première méthode, a une taille permettant d'atteindre une solution optimale en temps de calcul raisonnable. Au contraire, les problèmes peu contraints se prêtent bien à une résolution plus rapide par la deuxième méthode. Néanmoins, ces méthodes ne sont pas capables de résoudre les problèmes de grande taille (plus de 100 opérations) sans imposer un temps de calcul trop important. Il est à noter aussi que la deuxième approche s'est avérée tout de même plus sensible à l'augmentation du nombre d'opérations.

Ces résultats ne sont pas inattendus. Les problèmes d'équilibrage des lignes d'usinage sont NP-difficiles (Dolgui et al., 1999), il est donc illusoire de penser que les méthodes exactes peuvent traiter tous les problèmes, quelle que soit leur taille. Il est nécessaire alors de recourir à des méthodes approchées pour la résolution des instances de grande taille.

Quelques heuristiques ont été proposées dans (Finel, 2004; Dolgui et al., 2005; Dolgui et al., 2006a; Guschinskaya et al., 2007), dont la plus performante à ce jour, selon l'étude présentée dans (Guschinskaya et Dolgui, 2006), est l'approche mixte qui utilise l'heuristique FSIC (First Satisfy Inclusion Constraints) pour la construction des solutions admissibles et un algorithme de décomposition dynamique pour le découpage de telles solutions en sous-problèmes, qui sont ensuite résolus par une méthode exacte. Cependant, l'heuristique FSIC choisit une opération de la liste de candidats non pas en fonction d'une priorité, mais de manière aléatoire. Dans cet article, nous développons une nouvelle heuristique, baptisée GLB (pour Greedy Blocks Loading), qui, à la différence de l'heuristique FSIC, s'appuie sur une règle de priorité lors de la sélection des opérations à affecter. L'utilisation de ce nouvel algorithme heuristique nous permettra de mettre en place une méthode de type GRASP.

3. METHODE GRASP

La métaheuristique GRASP (Greedy Randomized Adaptive Search Procedure) a été proposée initialement par Feo et Resende (Feo et Resende, 1989). Elle fait partie des méthodes approchées qui sont applicables pour la résolution d'une large gamme de problèmes d'optimisation combinatoire. D'ailleurs, elle a été déjà appliquée avec succès à un grand nombre de problèmes d'optimisation. Des bibliographies sur GRASP ont été publiées par (Festa et Resende, 2001; Resende et Ribeiro 2003). La flexibilité de cette approche réside dans le fait que ses opérateurs de base ne sont pas axés sur un problème particulier, mais peuvent être adaptés à la résolution de problèmes comportant des structures différentes. Ainsi le développement d'une méthode de type GRASP consiste à choisir les opérateurs de base qu'elle comprend et les adapter au contexte du problème étudié.

Une méthode de type GRASP est constituée des deux phases suivantes : (1) construction d'une solution admissible à l'aide d'un algorithme glouton aléatoire, dit semi-glouton, (2) son amélioration par une méthode de recherche locale. Dans le schéma de GRASP, initialement proposé dans (Feo et Resende, 1989), ces deux phases sont répétées de manière itérative un grand nombre de fois jusqu'à ce qu'une condition

d'arrêt soit satisfaite. Afin de compléter l'approche de base utilisée par GRASP, de nombreux opérateurs d'approfondissement ont été proposés. En outre, de nombreuses hybridations avec d'autres métaheuristiques ont été mises en œuvre, pour plus de détails voir (Resende et Ribeiro 2003; Delorme, 2003). Une présentation générale de GRASP a été faite dans (Pitsoulis et Resende, 2002).

Dans cet article, nous adaptons l'approche par reactive GRASP au problème d'équilibrage des lignes de transfert. L'approche que nous mettons en œuvre est constituée des deux phases de base, proposées dans (Feo et Resende, 1989), et est dotée d'un seul opérateur d'approfondissement qui permet de piloter l'importance de la composante aléatoire lors de la construction de solutions initiales. La phase de construction est détaillée dans 3.1. La phase d'amélioration est considérée dans 3.2.

3.1. Phase de construction gloutonne aléatoire : heuristique GBL

Durant la phase de construction, une solution admissible doit être construite par une heuristique en utilisant une fonction gloutonne, que nous désignons comme $g(j)$. Un tel algorithme construit la solution élément par élément. À chaque étape de la construction, un ensemble *CL* des éléments-candidats, pouvant être ajoutés dans la solution, est construit. La valeur d'une fonction gloutonne est calculée pour chaque élément-candidat. Ensuite, les éléments sont classés dans l'ordre des valeurs de la fonction gloutonne (la direction de tri dépend de la fonction gloutonne choisie, qui peut être à minimiser ou à maximiser). L'élément le mieux placé est ajouté dans la solution. Le changement de la solution entraîne le changement du contenu de l'ensemble *CL*, qui est mis à jour, et le changement des valeurs de la fonction gloutonne pour ses éléments. Cependant, si l'algorithme utilisé ne comporte pas de composante aléatoire, la solution construite à chaque itération sera identique.

Pour introduire le facteur « hasard » et générer de différentes solutions de départ, sans pour autant perdre la manière rigoureuse de construction des solutions, on utilise des algorithmes semi-gloutons (Hart et Shogan, 1987; Feo et Resende, 1989). Un tel algorithme construit une liste complémentaire des éléments sélectionnés de la liste *CL*, cette liste est désignée comme *RCL* (Restricted Candidate List). Ensuite, un élément à affecter est choisi au hasard dans la liste *RCL*.

Différentes techniques de la construction de cette liste ont été proposées dans la littérature (Resende et al., 2000). Nous avons opté pour l'utilisation de la méthode suivante : un élément est placé dans la liste *RCL* si la valeur de la fonction gloutonne qui lui est attribuée n'est pas très loin de la meilleure valeur obtenue parmi les éléments de la liste *CL*. Dans ce qui suit, nous présentons l'heuristique GBL que nous avons développée. Tout d'abord, l'Algorithme 1 décrit son schéma général. Ensuite, chaque procédure de cet algorithme est énoncée, notamment les procédures de la construction des listes *CL* et *RCL*.

L'algorithme commence par l'ouverture d'un poste de travail et d'un seul bloc vide appartenant à ce poste. Soit m l'indice du poste courant et n_m l'indice du bloc courant du poste m. L'algorithme s'arrête soit lorsque toutes les opérations sont affectées, soit lorsqu'il reste encore des opérations non affectées, mais qu'aucune action n'est possible : ni de les affecter aux blocs et postes déjà existant, ni d'ouvrir un nouveau poste, car $m + 1 > m_0$. Dans le deuxième cas, l'itération courante est jugée non concluante et $C_{cur} = \infty$.

3.1.1. Construction de la liste *CL* de candidats

La liste N^{na} des opérations non affectées est analysée, l'opération j est placée dans la liste *CL* si elle peut être affectée au bloc courant, c'est-à-dire si toutes les conditions suivantes sont validées :

1. tous ses prédécesseurs sont déjà affectés, c'est-à-dire si $(i, j) \in D^{pr}$ alors $i \in \bigcup_{k=1}^{m} \bigcup_{l=1}^{n_m} N_{kb}$;
2. son affectation au bloc courant n'enfreint par la contrainte de temps de cycle pour le poste courant, c'est-à-dire :
   $t^S(N_m \setminus N_{mn_m}) + t^b(N_{mn_m} \cup \{j\}) \leq T_0$;
3. l'opération j n'est pas liée par des contraintes E^p avec les opérations déjà affectées au poste courant c'est-à-dire que $\forall e \in E^p$ tels que $j \in e$: $e \cap (N_m \cup \{j\}) \neq e$
4. l'opération j n'est pas liée par des contraintes E^b avec les opérations déjà affectées au bloc courant c'est-à-dire que $\forall e \in E^b$ tels que $j \in e$: $e \cap (N_{mn_m} \cup \{j\}) \neq e$.

Si $CL = \emptyset$, cela signifie qu'aucune opération parmi celles non encore affectées ne peut être assignée au bloc courant. Dans ce cas, il convient d'essayer d'ouvrir un nouveau bloc et de reconstruire la liste *CL*. Si l'ouverture d'un nouveau bloc au poste courant n'est pas possible ou si la liste *CL* est de nouveau vide et si, en même temps, le nombre de postes ouverts est inférieur à m_0, alors un nouveau poste est ouvert et la liste *CL* est reconstruite.

3.1.2. Construction de la liste *DAO*(*CL*)

Grâce à l'exécution parallèle d'opérations dans un

Initialiser $m = 1, n_m = 1, C_{cur} = C_1 + C_2$, $N^{na} = \mathbf{N}$, $newst = faux$
tant que $N^{na} \neq \emptyset$ **faire**
  si non *newst* **alors**
    Construire la liste *CL*
  fin
  si $CL = \emptyset$ **ou** *newst* **alors**
    si non *newst* **et** $n_m + 1 \leq n_0$ **alors**
      Set $n_m = n_m + 1$, $C_{cur} = C_{cur} + C_2$
    sinon
      si $m + 1 \leq m_0$ **alors**
        Set $m = m + 1$, $n_m = 1$,
        $C_{cur} = C_{cur} + C_1 + C_2$, $newst = faux$
      sinon
        Set $C = \infty$, **Quitter**.
      fin
    fin
  sinon
    Construire *DAO*(*CL*).
    si *DAO*(*CL*) $\neq \emptyset$ **alors**
      Affecter tous les $j \in$ *DAO*(*CL*)
    sinon
      Set $assign = faux$
      tant que $assign \neq vrai$ **faire**
        Construire *RCL*
        Choisir l'opération i de la liste *RCL*.
        Construire *AL*.
        si $|AL| > 1$ **alors**
          Set $S_{copy} = S_{cur}$
        fin
        Set $assign =$ *ASSIGN*(*AL*)
        si $assign = vrai$ **alors**
          $N^{na} = N^{na} \setminus AL$
        sinon
          Set $S_{cur} = S_{copy}$,
          $RCL = RCL \setminus \{i\}$, $CL = CL \setminus \{i\}$
          si $CL = \emptyset$ **alors**
            si $N_{mn_m} = \emptyset$ **alors**
              Set $n_m = n_m - 1$,
              $C_{cur} = C_{cur} - C_2$,
              $newst = vrai$
            fin
            Quitter
          fin
        fin
      fin
    fin
  fin
fin

Algorithme 1: Algorithme GLB

bloc, certaines opérations se font en temps caché, c'est-à-dire que les paramètres de deux opérations au plus sont utilisés pour le calcul de temps d'usinage. Il en découle qu'il est possible d'affecter des opérations à un bloc contenant déjà des opérations sans changer son temps. Ceci est réalisable pour chaque opération j pour laquelle :

$$\begin{array}{l} L(N_{mn_m}) \geq l_j, \\ v_f(N_{mn_m}) \geq v_{f1}(j), \\ v_f(N_{mn_m}) \leq v_{f0}(j), \\ v_f(N_{mn_m}) \leq v_{f2}(j) \end{array} \quad (11)$$

Si la condition (11) n'est pas valide, nous pouvons vérifier les conditions suivantes :

$$\begin{array}{l} \min\{l_i \mid i \in CL, i \neq j\} \geq l_j, \\ \min\{v_{f0}(i) \mid i \in CL, i \neq j\} \geq v_{f1}(j), \\ \max\{v_{f0}(i) \mid i \in CL, i \neq j\} \leq v_{f0}(j), \\ \max\{v_{f0}(i) \mid i \in CL, i \neq j\} \leq v_{f2}(j) \end{array} \quad (12)$$

Si pour l'opération j la condition (11) ou (12) est respectée et si en plus cette opération n'est liée avec aucune opération non affectée par une contrainte d'exclusion ou d'inclusion, alors, nous pouvons constater que l'affectation de l'opération j au bloc courant n'exclut aucune opération de la liste *CL*. Dans le cas où la condition (12) est vérifiée, nous pouvons constater que même si le temps mort du poste courant est réduit après l'affectation de l'opération j, toutes les autres opérations restent encore affectables au bloc courant.

Par conséquent, nous pouvons affecter de telles opérations avant les autres sans exclure aucune opération de la liste *CL*. Soit *DAO*(*CL*) (Directly Assigned Operations) la liste de telles opérations dans la liste *CL*. Si *DAO*(*CL*) $\neq \emptyset$, alors toutes les opérations $j \in$ *DAO* sont affectées au bloc courant et seulement après la liste *CL* est reconstruite. La même procédure est répétée jusqu'à *DAO*(*CL*) $= \emptyset$. Si *DAO*(*CL*) $= \emptyset$, alors l'algorithme passe à l'étape suivante.

3.1.3. Évaluation gloutonne des opérations

La différence principale entre les heuristiques FSIC et GLB se situe à cette étape. Si l'heuristique FSIC choisit l'opération j de la liste *CL* de manière aléatoire, l'heuristique GLB, elle, utilise une règle pour effectuer ce choix.

Cette règle a pour objectif de choisir les affectations qui mènent à une solution finale de bonne qualité, c'est-à-dire ayant le moindre coût. Évidement, car toutes les possibilités ne seront pas exploitées, l'optimalité de la solution obtenue ne peut pas être garantie, mais une solution de bonne qualité devrait être atteinte plus rapidement qu'avec l'heuristique FSIC. La difficulté réside dans le développement de cette règle. Nous avons utilisé la règle suivante : la borne inférieure sur le nombre de blocs nécessaires pour affecter tous les successeurs de l'opération j. Évidement, plus la valeur de cette borne est grande et plus de blocs seront ouverts après l'affectation

de l'opération j. Par conséquent, plus tôt il faut l'affecter. En outre, si plusieurs opérations ont les mêmes valeurs, nous choisissons l'opération qui est liée par des contraintes d'exclusion avec d'autres opérations ayant la même valeur de la borne.

3.1.4. Construction de la liste *RCL*

Nous avons opté pour l'utilisation de la méthode suivante pour la construction de la liste *RCL* : un élément y est placé, si la valeur de la fonction gloutonne qui lui est attribuée n'est pas très loin de la meilleure valeur obtenue pour les éléments de la liste *CL* :

$$RCL = \{j \in CL | g(j) \geq g_{max} - \alpha(g_{max} - g_{min})\}$$

où g_{max} et g_{min} représentent les valeurs maximum et minimum de la fonction g pour les éléments de la liste *CL*.

Des expérimentations préliminaires ne nous ont pas permis d'identifier une valeur particulière du paramètre α pouvant être considérée comme une recommandation pour la résolution de toutes les instances (ou au moins une grande partie). En conséquence, nous avons décidé de mettre en place la procédure d'ajustement de la valeur du paramètre α lors de la résolution de chaque problème particulier. De cette manière, la valeur utilisée est à chaque fois adaptée aux particularités du problème à résoudre. La procédure d'ajustement opère avec un ensemble discret des valeurs de α pré-définies (cet ensemble est désigné comme αSet). Chaque valeur de l'ensemble αSet a une probabilité d'être choisie par l'algorithme, désignée par pr_α. Au début de la première itération, toutes les valeurs ont la même probabilité d'être choisies, égale à :

$$pr_\alpha = \frac{1}{|\alpha Set|}, \forall\, \alpha \in \alpha Set$$

Mais au fur et à mesure des itérations, ces probabilités sont modifiées afin de privilégier le choix des valeurs de α qui amènent à des meilleures solutions finales. Les probabilités du choix des valeurs prédéfinies $\alpha \in \alpha Set$, pr_α sont mises à jour selon une périodicité dépendant de la condition *prUpdate*. Les nouvelles probabilités sont calculées à partir du coût moyen des meilleures solutions obtenues en utilisant chaque valeur du paramètre α (ces solutions sont stockées dans les ensembles S_α) en tenant compte du meilleur et du pire coût parmi toutes les solutions déjà obtenues, représentées par l'ensemble de solutions $\mathbf{S}_{gr}$.

Tout d'abord, la valeur de val_α est calculée pour chaque $\alpha \in \alpha Set$ de la manière suivante :

$$\begin{array}{l} val_\alpha = [(\max\{\, C(S) \mid S \in \mathbf{S}_{gr}\} - \\ -\text{moyen}\{\, C(S) \mid S \in S_\alpha\})/ \\ (\max\{\, C(S) \mid S \in \mathbf{S}_{gr}\} - \\ -\min\{\, C(S) \mid S \in \mathbf{S}_{gr}\})]^\sigma \end{array} \quad (13)$$

où le paramètre σ est un paramètre de contrôle qui est utilisé afin d'atténuer les écarts entre les probabilités des différentes valeurs du paramètre α. La taille maximale des ensembles S_α est un paramètre de contrôle de l'algorithme.

Ensuite, la probabilité du choix de la valeur du paramètre $\alpha \in \alpha Set$ est calculée comme suit :

$$pr_\alpha = \frac{val_\alpha}{\sum_{\alpha \in \alpha Set} val_\alpha} \quad (14)$$

L'Algorithme 2 récapitule l'approche GRASP réactive que nous avons mis en place pour la résolution du problème d'équilibrage des lignes de transfert.

3.1.5. Construction de la liste *AL*

Rappelons qu'afin de diminuer le nombre de solutions inadmissibles, nous devons veiller à l'affectation de toutes les opérations liées par une contrainte d'inclusion au même poste de travail. Pour cela, si l'opération j, qui a été choisie pour être affectée, possède ce genre des contraintes ($OMP(j) \neq \emptyset$), alors l'algorithme crée une liste complémentaire *AL* contenant toutes les opérations de $OMP(j)$ non affectées et tous leurs prédécesseurs non affectés. En effet, nous ne pouvons pas, en général, savoir d'avance le nombre de blocs nécessaire pour l'affectation de toutes les opérations de *AL*, et nous ne pouvons pas savoir s'il sera possible de toutes les affecter au poste courant. Pour cette raison, l'état de la solution courant S_{cur} (désigné par S_{copy}) est sauvegardé avant l'affectation de la première opération de la liste *AL*. Ensuite, les opérations comprises dans la liste *AL* sont affectées de la même façon que les opérations de la liste *CL*, à savoir les valeurs de la fonction gloutonne sont calculées pour toutes les opérations, puis la liste *RCL* est construite et l'opération à affecter est sélectionnée. Les opérations de la liste *AL* sont traitées en priorité. De cette façon, nous recourons à la liste *CL* pour choisir une opération à affecter au bloc courant seulement si aucune opération de la liste *AL* ne peut plus y être affectée. Bien évidemment, en aucun cas, nous ne pouvons choisir une opération qui, par ses contraintes d'exclusion, empêcherait l'affectation des opérations de la liste *AL* au poste courant.

Si toutes les opérations de la liste *AL* ont été affectées avec succès au poste courant, alors la liste *CL* est

Initialiser $C_{min} = \infty$, $TR_{tot} = 0$, $TR_{nimp} = 0$,
$pr_\alpha = \frac{1}{|\alpha Set|} \; \forall \, \alpha \in \alpha Set$.
répéter
  Choisir α de l'ensemble αSet en tenant compte de Pr_α
  Construction d'une solution admissible S_{cur} ayant le coût C_{cur}
  si $C_{cur} < C_{min}$ **alors**
    $C_{min} = C_{cur}$, $S_{min} = S_{cur}$
  fin
  si $C_{cur} < \infty$ **alors**
    Phase d'amélioration
    si $C_{cur} < C_{min}$ **alors**
      $C_{min} = C_{cur}, S_{min} = S_{cur}$, $TR_{nimp} = 0$.
    fin
  fin
  si $C_{cur} \geq C_{min}$ **alors**
    $TR_{nimp} = TR_{nimp} + 1$
  fin
  si $prUpdate$ **alors**
    $val_{\sum \alpha} = 0$
    pour chaque $\alpha \in \alpha Set$ **faire**
      Calculer val_α avec ()
      $val_{\sum \alpha} = val_{\sum \alpha} + val_\alpha$
    fin
    pour chaque $\alpha \in \alpha Set$ **faire**
      $pr_\alpha = \frac{val_\alpha}{val_{\sum \alpha}}$
    fin
  fin
  $TR_{tot} = TR_{tot} + 1$
jusqu'à $T_{cur} > T_{res}$ **ou** $TR_{nimp} > TR_{nimp\_aut}$ **ou** $TR_{tot} > TR_{tot\_aut}$ **ou** $C_{min} < C_{stop}$

Algorithme 2: Schéma général de la GRASP réactive

mise à jour et la construction de la solution se poursuit, sinon l'algorithme restaure l'état de la solution avant l'affectation de la première opération de la liste *AL* : $S_{cur} = S_{copy}$. L'opération j est ensuite supprimée des listes *RCL* et *CL*. Puis, une autre opération est sélectionnée de la liste *RCL*.

3.2. Phase d'amélioration par recherche locale

La métaheuristique GRASP recoure à une procédure de recherche locale pour améliorer les solutions obtenues à la phase de construction. Compte tenu de bons résultats que nous avons obtenus en utilisant l'approche de décomposition dynamique d'une solution heuristique par son découpage en sous-problèmes pour l'approche mixte, nous avons décidé de l'intégrer dans notre approche GRASP à la phase d'amélioration de solutions. Le voisinage que nous considérons est donc défini par l'ensemble des recompositions possibles de la solution après un découpage aléatoire en sous-problèmes. Il est intéressant de noter que, du fait du caractère aléatoire du découpage, l'application de la recherche locale plusieurs fois à la même solution initiale ne conduira pas nécessairement au même optimum local.

Dans un premier temps, l'heuristique est utilisée afin de trouver une solution admissible du problème initial S_{heur} qui consiste en une affectation possible de toutes les opérations de l'ensemble **N** aux m_{heur} postes de travail. La solution trouvée par l'heuristique GLB peut être représentée comme une séquence S_{heur}, où chaque élément correspond à un poste de travail. Bien évidemment, toutes les contraintes de précédence et de compatibilité sont respectées pour chaque élément, sur chaque poste de travail et entre eux. Pour améliorer cette solution, nous utilisons la procédure présentée par la figure 1.

Cette procédure consiste à découpe la solution heuristique en sous-ensembles indépendants. Ceci est mis en œuvre par un découpage de la séquence S_{heur} en w sous-séquences, de façon à obtenir la décomposition suivante $S_{heur} = \{N^1, ..., N^u, ..., N^w\}$, dont les sous-ensembles N^u respectent les conditions suivantes :

- chaque sous-ensemble N^u inclut un nombre entier de postes de travail successifs, c'est-à-dire $N^u = \bigcup_{k=k_1}^{k_2} N_k, 1 \leq k_1 < k_2 \leq m_{heur}$;
- toutes les opérations de chaque poste de travail sont dans le même sous-ensemble, c'est-à-dire si $N_k \subset N^u$, alors $N_k \not\subset N^i, i \in \{1, ..., w\} \backslash u$;
- l'union de tous les sous-ensembles N^u donne l'ensemble **N**, c.-à-d. $\bigcup_{u=1}^{w} N^u = \mathbf{N}$.

Comme nous ne disposons pas de critère permettant de déterminer a priori le découpage en sous-séquences qu'il faut utiliser pour avoir plus de gain, nous effectuons le découpage de manière aléatoire. Plus précisément, pour déterminer le nombre de postes de travail dont les opérations constitueront un sous-problème (k_u), nous choisissons au hasard un nombre entier dans l'intervalle [1 .. $MaxSt$], où le paramètre $MaxSt$ comme $MaxOp$ est utilisé pour limiter la taille des sous-problèmes : il donne le nombre maximal autorisé de postes de travail inclus dans un sous-problème. Le choix des paramètres $MaxOp$ et $MaxSt$ doit être fait à l'étape d'initialisation de l'algorithme en fonction des paramètres suivants : la taille du problème initial, le temps de calcul alloué, la technique de formation de sous-problèmes et la méthode exacte utilisée pour leur résolution. Si lors de l'exécution de l'algorithme, ces deux paramètres sont en contradiction, c'est le paramètre $MaxOp$ qui domine.

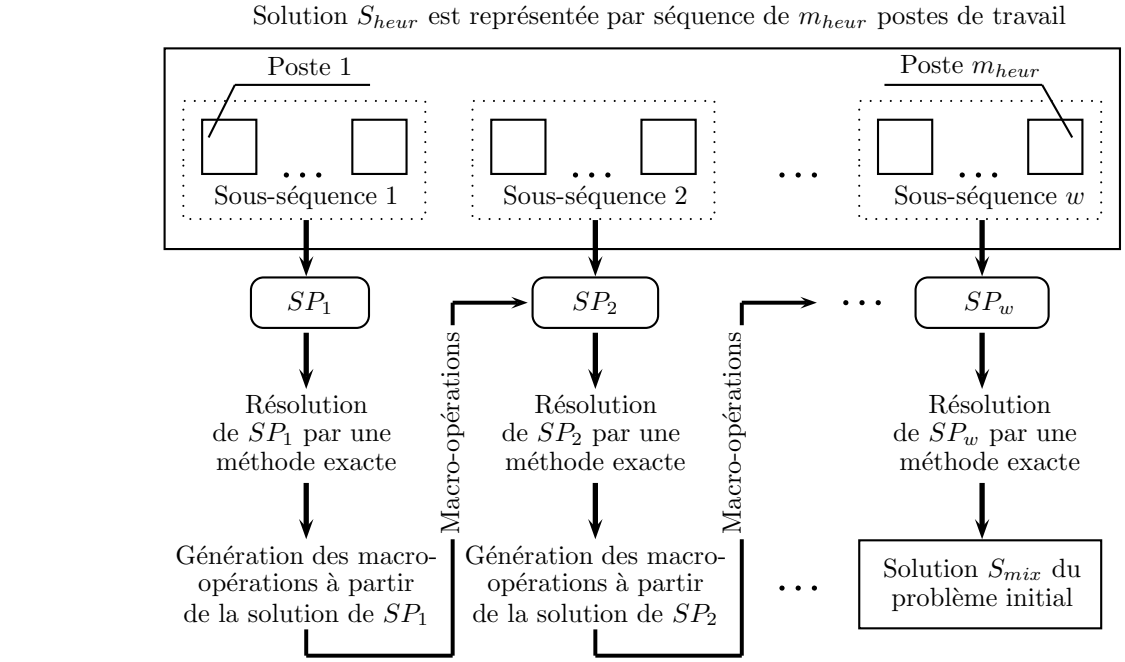

Figure 1: Schéma de la décomposition

Ensuite, à partir de chaque sous-ensemble N^u, obtenu par le découpage de la séquence S_{heur}, nous formons un nouveau problème, dit sous-problème, qui est de même type que le problème initial mais de taille plus petite, il inclut les opérations du sous-ensemble N^u et les opérations qui étaient dans les sous-problèmes précédents N^r, où $r = 1, ..., u-1$, mais sous forme de macro-opérations. Ainsi la solution du problème initial est le résultat de la résolution du dernier sous-problème.

Ainsi la solution améliorée est construite en augmentant progressivement la taille du sous-problème de telle façon que les opérations qui ont données lieu aux sous-problèmes antérieurs soient également inclues dans le sous-problème courant. Ainsi, la solution du problème initial est le résultat de la résolution du dernier sous-problème.

Après avoir résolu le sous-problème SP_{u-1}, il suffit de remplacer les blocs obtenus dans la solution S_{u-1} par des macro-opérations. Avec cette stratégie, nous tenons compte de la possibilité d'ajouter dans les postes de travail et blocs déjà construits les opérations qui n'étaient pas traitées au moment de la formation de ces blocs et postes, ainsi que la possibilité d'assigner les blocs existants aux autres postes de travail que ceux auxquels ils ont été assignés auparavant.

Nous utilisons la procédure suivante pour créer les macro-opérations. Nous remplaçons chaque ensemble d'opérations N_{kb} affecté au b-éme bloc du poste k dans la solution S_{u-1} par une seule macro-opération MO ayant le numéro $num(MO)$. Les paramètres $l_{num(MO)}$, $v_{f1}(num(MO))$, $v_{f0}(num(MO))$, $v_{f2}(num(MO))$ de cette macro-opération sont calculés à partir des valeurs individuelles des opérations de l'ensemble N_{kb}. Après avoir créé toutes les macro-opérations, nous ajustons les contraintes du problème initial de la manière suivante : si N_{kb} est un élément de I^p, alors cet élément peut être supprimé, car la contrainte correspondante est déjà respectée ; autrement, s'il existe des opérations appartenant à la fois à l'ensemble N_{kb} et à des ensembles des collections E^b, E^p ou I^p (mais N_{kb} n'est pas un élément de I^p), alors nous les remplaçons par les numéros des macro-opérations correspondantes.

Les macro-opérations sont ajoutées dans l'ensemble N^u. Le paramètre m_0 prend une nouvelle valeur $m_0 = \lfloor (C_{u-1} + C_u^{heur})/(C_1 + C_2) \rfloor$, où C_{u-1} est le coût de la solution du sous-problème SP_{u-1}.

Notons que nous devons garder les informations concernant la composition des macro-opérations, car pour reconstituer la solution du problème initial après avoir obtenu la solution du dernier sous-problème, il faut remplacer les macro-opérations dans cette solution par les opérations d'origine.

3.2.3. Résolution des sous-problèmes

Suite au découpage aléatoire du problème initial, nous pouvons nous retrouver face à deux situations problématiques lors de la résolution d'un sous-problème. Malgré l'utilisation des paramètres de contrôle de la taille, le sous-problème formé peut s'avérer « trop lourd » pour la méthode exacte utilisée à cause de sa structure. Si nous tentons de résoudre un tel sous-problème, nous risquerons de consommer tout le temps de résolution alloué rien que pour ce sous-

problème et ne pas avoir le temps pour les autres sous-problèmes, ce qui ne garantit pas l'amélioration de la solution finale. Notons que le terme « trop lourd » dépend de la méthode exacte utilisée. Si la taille d'un sous-problème est « trop petite », alors, la solution heuristique partielle a une forte chance d'être optimale, et par conséquent, la résolution de ce sous-problème de manière exacte ne pourra pas l'améliorer. Par expérience, nous qualifions la taille d'un sous-problème comme « trop petite » si le sous-problème contient un seul poste de travail avec un nombre de blocs inférieur à 3, ou deux postes de travail avec un nombre de blocs inférieur à 2 chacun.

Alors, si à l'étape de résolution d'un sous-problème SP_u, nous nous apercevons que sa taille est « trop petite » ou « trop grande », afin d'éviter les pertes du temps, nous n'appliquons pas la méthode exacte, mais nous copions simplement la sous-séquence u dans la solution améliorée.

De plus, afin d'éviter la résolution des sous-problèmes pour lesquels les solutions optimales sont déjà connues, nous calculons la borne inférieure du coût pour chaque sous-problème. Si jamais le coût de la solution heuristique est égal à la borne inférieure, alors il est inutile de résoudre ce sous-problème.

Toute méthode exacte peut être utilisée pour la résolution de sous-problèmes, nous utilisons celle proposée dans (Dolgui et al., 2008).

4. ÉVALUATION DE LA MÉTHODE

Afin d'étudier les performances de la méthode GRASP, nous avons créé un échantillon de problèmes générés aléatoirement, mais en tenant compte des caractéristiques des problèmes industriels.

La pièce à fabriquer a généralement une multitude d'éléments fonctionnels tels que les différentes rainures, trous, saillies, etc. La description de la pièce est souvent définie par l'intermédiaire du concept d'entité. L'entité est un groupement sémantique des opérations, on peut distinguer plusieurs types d'entités selon les opérations qu'elles comprennent. Chaque type est caractérisé par un ensemble d'opérations définies de natures différentes (fraisage, perçage, alésage, etc.) et par les contraintes entre elles. Ainsi, la fabrication d'une pièce consiste à usiner un ensemble d'entités technologiques.

Lors de la génération des instances à tester, nous avons utilisé de vraies entités. Plus il y a d'entités, plus la taille de l'ensemble **N** est grande. Alors, afin d'étudier l'impact du nombre d'opérations sur la qualité des solutions obtenues, nous avons créé 4 séries de tests. Chacune est caractérisée par le nombre d'entités qui ont été utilisées lors de la génération des instances.

Chaque série comporte 50 instances distinctes qui ont été générées de manière aléatoire, au total, nous avons donc 200 instances. Le type de chaque entité et la face de la pièce à laquelle cette entité appartient ont été choisis au hasard. De cette façon, les instances de la même série ont des nombres d'opérations et des ensembles de contraintes différents.

Le Tableau 1 fournit les paramètres des séries, à savoir : « Série » représente le numéro de la série, n_{ent} est le nombre d'entités utilisé lors de la génération des instances, $|N_{min}|, |N_{max}|, |N_{moy}|$ sont les valeurs minimum, maximum et moyennes du nombre d'opérations pour les instances de la série correspondante.

Série	n_{ent}	$\|N_{min}\|$	$\|N_{max}\|$	$\|N_{moy}\|$
1	10	29	47	38
2	20	46	92	55
3	30	80	127	84
4	40	115	158	141

Table 1: Paramètres des séries de tests

Notre but est d'évaluer les performances de la méthode GRASP en la comparant avec l'approche mixte. Tous les instances de 4 séries ont été résolus par les 2 méthodes. Les résultats fournis par chaque méthode pour chaque problème ont été comparés afin de trouver la meilleure solution obtenue. Ensuite, les écarts par rapport au meilleur résultat ont été calculés, ils sont représentés dans le tableau 2 par Δ_{min}, Δ_{moy} et Δ_{max} qui sont respectivement l'écart minimum, moyen et maximum, *PMS* est le Pourcentage des Meilleures Solutions. Les calculs ont été effectués sur une machine ayant un Pentium IV, 3GHz et 512 Mo de RAM. Les résultats obtenus sont présentés dans le Tableau 2.

Série	Méthode	Δ_{min}	Δ_{max}	Δ_{av}	*PMS*
1	Mixte	0	4,76	0,3	93,3
	GRASP	0	0	0	100
2	Mixte	0	6	1,62	39,13
	GRASP	0	0	0	100
3	Mixte	0	11,2	3,5	14,3
	GRASP	0	0	0	100
4	Mixte	0,78	11,76	5,43	0
	GRASP	0	0	0	100

Table 2: Résultats de la comparaison des approches mixte et GRASP

5. CONCLUSION

Nous avons proposé une méthode de type GRASP pour l'équilibrage des lignes de transfert avec des

postes de travail comportant plusieurs boîtiers multi-broches. Le problème consiste à minimiser le coût de la ligne tout en assurant le taux de productivité voulu et en respectant toutes les contraintes données. Les résultats obtenus nous permettent de conclure que l'utilisation de l'heuristique GBL à la phase de construction de solutions améliore considérablement la qualité des résultats finaux. Nous pouvons même constater que l'approche GRASP surpasse incontestablement l'approche mixte. Cette supériorité se révèle particulièrement importante lors de la résolution des problèmes de grande taille. Par exemple, pour toute instance de la série 4, l'approche GRASP a donné un résultat meilleur que l'approche mixte. L'écart entre les deux solutions est en moyenne de 5,43% pour la méthode mixte. Cette hypothèse est également confirmée par l'analyse des valeurs du paramètre *PMS*. Pour les instances de petite taille, le pourcentage des meilleures solutions trouvées par l'approche mixte est proche de 100% ; ensuite, il diminue avec l'augmentation de la taille d'instances jusqu'à ce qu'il soit carrément nul pour les instances de grande taille. Nous expliquons ce résultat par l'aptitude de l'heuristique GLB à fournir de bonnes solutions de départ. Ceci permet d'obtenir de bonnes solutions finales avec un temps relativement court de résolution.

REFERENCES

Delorme, X. 2003. *Modélisation et résolution de problèmes liés à l'exploitation d'infrastructures ferroviaires*, Thèse de doctorat. Université de Valenciennes et du Hainaut-Cambrésis.

Dolgui, A., B. Finel, N. Guschinsky, G. Levin, and F.Vernadat, 2005. A heuristic approach for transfer line sbalancing. *Journal of Intelligent Manufacturing*, 16(2), p. 159-171.

Dolgui, A., B. Finel, O. Guschinskaya, N. Guschinsky, G. Levin, and F. Vernadat, 2006a. Balancing large-scale machining lines with multi-spindle heads using decomposition. *International Journal of Production Research*, 44(18-19), p. 4105-4120.

Dolgui, A., B. Finel, N. Guschinsky, G. Levin, and F. Vernadat, 2006b. MIP approach to balancing transfer lines with blocks of parallel operations, *IIE Transactions*, 38, p. 869–882.

Dolgui, A., N. Guschinsky and G. Levin, 1999. Optimal design of transfer lines and multi-position machines. In: *Proceedings of the 7th IEEEMediterranean Conference on Control and Automation.* ETFA'99, Haifa, Israel. p. 1962-1973.

Dolgui, A., N. Guschinsky, G. Levin, and J. Proth, 2008. Optimisation of multi-position machines and transfer lines, *European Journal of Operational Research*,185(3), p. 1375–1389.

Feo, T. and M. Resende, 1989. A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters*, 8, p. 67–71.

Festa, P. and M. Resende, 2001. GRASP: An annotated bibliography, *in* C. Ribeiro et P. Hansen (eds), *Essays and Surveys on Metaheuristics*, Kluwer Academic Publishers, p. 325–367.

Finel, B., 2004. *Structuration de lignes d'usinage : méthodes exactes et heuristiques.* Thèse de doctorat. Université de Metz.

Guschinskaya, O. and A. Dolgui, 2006. A comparative evaluation of exact and heuristic methods for transfer lines balancing problem, *in* A. Dolgui, G. Morel and C. E. Pereira (eds), *Information Control Problems In Manufacturing* 2006 : *A Proceedings volume from the 12th IFAC International Symposium*, Vol. 2, Elsevier Science, p. 395–400.

Guschinskaya,O., A. Dolgui, N. Guschinsky, and G. Levin, 2007. A heuristic multi-start decomposition approach. *European Journal of Operational Researc*h. À paraître, doi:10.1016/j.ejor.2006.03.072.

Hart, J. and A. Shogan, 1987. Semi-greedy heuristics: An empirical study, *Operations Research Letters*, 6, p. 107–114.

Pitsoulis, L. and M. Resende, 2002. Greedy randomized adaptive search procedures, *in* P. Pardaloset and M. Resende (eds), *Handbook of Applied Optimization*, Oxford University Press, New York, p. 168–183.

Resende, M., L. Pitsoulis, and P. Pardalos, 2000. Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP, *Discrete Applied Mathematics*, 100, p. 95–113.

Resende, M. and C. Ribeiro, 2003. Greedy randomized adaptive search procedures, *in* F. Glover and G. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, p. 219–249.