Approche Multi-Agents basée sur la Recherche Tabou pour le Job Shop flexible

Multi-Agent Approach based on Tabu search for the flexible job shop scheduling problem

Meriem Ennigrou[1] — Khaled Ghédira[2]

1 ur. SOI²E, Stratégies d'Optimisation de l'Ingénierie des Informations et de la connaissancE
IPEIM, Institut Préparatoire aux Etudes d'Ingénieurs – ElManar
2092 El Manar 2 BP 244, Tunisie
meriem.ennigrou@enit.rnu.tn

2 ur. SOI²E, Stratégies d'Optimisation de l'Ingénierie des Informations et de la connaissancE
ENSI, Ecole Nationale des Sciences de l'Informatique
2010 Campus Universitaire la manouba, Tunisie
khaled.ghedira@isg.rnu.tn

Résumé

Ce papier propose une approche Multi-Agents utilisant la méthode de recherche tabou pour la résolution du problème d'ordonnancement d'atelier de type Job Shop flexible. La caractéristique de ce type de problème est qu'une opération peut être exécutée par un ensemble de ressources de telle sorte que sa durée de traitement dépend de la ressource utilisée. Cette généralisation du problème classique le rend de plus en plus difficile à résoudre. L'objectif de la résolution est la minimisation de la durée totale de l'ordonnancement ou makespan. Le modèle proposé est constitué de trois classes d'agents: les agents Job et les agents Ressource responsables de la satisfaction des contraintes sous leur responsabilité, et d'un agent Interface contenant le noyau de la méthode de recherche tabou. Différentes expérimentations ont été réalisées sur des benchmarks divers.

Mots Clef

Système Multi-Agents, Job Shop flexible, Ordonnancement, Recherche Tabou.

Abstract

This paper proposes a Multi-agent approach based on a tabu search method for solving the flexible Job Shop scheduling problem. The characteristic of the latter problem is that one or several machines can process one operation so that its processing time depends on the machine used. Such a generalization of the classical problem makes it more and more difficult to solve. The objective is to minimize the makespan or the total duration of the schedule. The proposed model is composed of three classes of agents: Job agents and Resource agents which are responsible for the satisfaction of the constraints under their jurisdiction, and an Interface agent containing the tabu search core. Different experimentations have been performed on different benchmarks.

Keywords

Multi-Agent System, flexible Job Shop, Scheduling, Tabu Search

1 Introduction

Les problèmes d'ordonnancement sont présents dans tous les secteurs de l'économie et constituent une fonction importante en gestion de production. Un problème d'ordonnancement consiste à allouer dans le temps des jobs à des ressources existantes en quantité limitée tout en satisfaisant un ensemble de production.

Parmi les problèmes d'ordonnancement les plus difficiles on cite le problème d'ordonnancement d'atelier de type *Job Shop*. Sa résolution de manière optimale s'avère dans la plupart des cas impossible à cause de son caractère fortement combinatoire. En effet, pour de tels problèmes, les méthodes exactes requièrent un effort calculatoire qui croît exponentiellement avec la taille du problème. Pour cela, des méthodes approchées ont été proposées pour résoudre ce type de problèmes en un temps raisonnable. Parmi ces méthodes, on mentionne celles qui sont basées sur le principe de la *recherche locale*, telles que *recherche tabou*, *recuit simulé*, etc., ou celles qui sont basées sur des méthodes évolutives, telles que *algorithmes génétiques*, *colonies de fourmis*, etc.

Dans ce papier, on présente un modèle Multi-Agents basé sur la méthode de recherche tabou pour la résolution du problème Job Shop flexible. Ce

problème constitue une généralisation du problème classique et est de ce fait plus difficile à résoudre.

Le papier est organisé comme suit : la section 2 définit le problème sur lequel on travaille, la section suivante présente la méthode de recherche tabou utilisée dans notre approche, la section 4 décrit le modèle Multi-Agents proposé ainsi que les agents qui le composent et sa dynamique globale, la section 5 illustre cette dynamique à travers un exemple. Enfin, la dernière section présente quelques résultats expérimentaux.

2 Le problème Job Shop flexible

Le problème Job Shop consiste à réaliser un ensemble de n jobs $\{J_1, ..., J_n\}$ sur un ensemble de m ressources $\{R_1, ..., R_m\}$. Chaque job J_i, $i=1, ..., n$, est composé d'une suite de n_i opérations devant être exécutées sur les différentes ressources selon un ordre préalablement défini. Cet ordre est appelé *gamme opératoire* et traduit les *contraintes de précédence* entre les différentes opérations du job. Par ailleurs, chaque opération possède une durée de traitement connue à l'avance et elle ne peut être exécutée que par une et une seule ressource. De plus, chaque job doit être réalisé dans un intervalle de temps déterminé par la date de début au plus tôt du job, ou *release date*, avant laquelle le traitement ne peut pas commencer, et par sa date de fin au plus tard, ou *due date*, déterminant une limite supérieure pour la fin de réalisation. Ces contraintes définissent les *contraintes temporelles* relatives au job. En outre, une ressource ne peut exécuter qu'une seule opération à un instant donné, ce qui définit les *contraintes disjonctives* des ressources, et une opération ne peut pas être interrompue une fois qu'elle est commencée. Une solution consiste alors à définir pour chaque opération une date de début satisfaisant l'ensemble des contraintes.

Le problème Job Shop flexible, introduit par [12], représente une généralisation du problème présenté précédemment, du fait qu'une opération donnée peut être réalisée par une ou plusieurs ressources et possède une durée de traitement dépendant de la ressource utilisée. L'ensemble des contraintes de ce problème est aussi composé des contraintes de précédence, contraintes temporelles et contraintes disjonctives. Une solution au problème Job Shop flexible ne consiste plus uniquement à trouver une séquence des opérations sur chacune des ressources et à leur fixer une date de début mais également à déterminer une affectation de chaque opération à l'une de ses ressources potentielles de telle sorte que la durée totale de l'ordonnancement est minimisée. Ce problème est NP- difficile puisque le problème classique est aussi NP-difficile [6].

Quelques approches ont été proposées pour la résolution de ce problème et basées sur la méthode de recherche tabou. Parmi ces dernières on cite à titre d'exemple celles proposées par [2], [3], [11], etc.

3 Recherche tabou

Le modèle proposé dans cet article est basé sur la méthode de recherche tabou [8] qui est une méta-heuristique fondée sur le principe de la recherche locale. Ce principe consiste à explorer l'espace de recherche composé de toutes les solutions réalisables dans le but d'aboutir à la solution optimale. Commençant à partir d'une solution initiale réalisable, le processus consiste, à chaque itération, à choisir la meilleure solution dans le voisinage de la solution courante, même si cette solution n'entraîne pas une amélioration. Ce voisinage est constitué de toutes les solutions obtenues en effectuant une seule *transition* sur la solution courante. Ces solutions sont alors appelées *voisins* de la solution courante.

Afin d'éviter le piège des optima locaux dans lequel ce processus peut être facilement attrapé, la recherche tabou utilise une structure de mémorisation temporaire dans laquelle elle sauvegarde les dernières solutions visitées : la *liste tabou*. En effet, une solution reste interdite pendant un nombre d'itérations égal à la taille de cette liste. Par suite, le meilleur voisin non tabou sera choisi pour la prochaine itération.

Bien que cette méthode a montré son efficacité pour la résolution de plusieurs problèmes difficiles, elle reste, néanmoins, difficile à adapter au problème Job Shop flexible. Ceci est surtout dû au nombre considérable de paramètres à définir :

- Solution initiale,
- fonction voisinage,
- évaluation de la solution courante,
- taille de la liste tabou, etc.

Dans ce qui suit, nous donnons brièvement l'adaptation des différents paramètres au problème Job Shop flexible avant de décrire, dans la section suivante, le modèle Multi-Agents proposé ainsi que sa dynamique globale.

3.1 Fonction voisinage

La complexité d'une approche de résolution basée sur la recherche taboue dépend essentiellement de (1) la taille du voisinage de la solution courante et (2) de la méthode d'évaluation de chacun de ces voisins afin de déterminer celui qui minimise la fonction coût. Il a été démontré par [5] que près de 90% du temps de résolution est pris par l'évaluation des voisinages. Par conséquent, il est intéressant d'utiliser des voisinages aussi contraints que possible afin de diminuer au maximum la complexité de résolution.

Avant de présenter la fonction voisinage utilisée, on commence par donner la définition de chemin critique.

Définition. Un *chemin critique* d'une solution est un chemin dont la longueur est égale à la longueur de l'ordonnancement et est constitué par des opérations reliées soit :

– Par un lien de précédence.

– Par un lien disjonctif (pouvant être réalisées par la même ressource).

Définition Une *opération critique* est une opération appartenant au chemin critique.

Le voisinage d'une solution S est alors obtenu par deux types de mouvements :

1. Permutation de deux opérations critiques adjacentes exécutées par la même ressource.
2. Replacement d'une opération critique sur une autre ressource pouvant l'exécuter.

3.2 Evaluation du voisinage

Le meilleur voisin non tabou parmi le voisinage de la solution courante sera sélectionné pour l'itération suivante. Pour cela, il faudrait pouvoir évaluer tous les voisins. Cependant, une évaluation complète, i.e. calcul de toutes les dates de début de toutes les opérations, de chaque voisin prend un temps considérable. Ceci nous a amené à calculer seulement les dates de début d'un sous-ensemble d'opérations qui sont effectivement concernées par le mouvement.

Dans ce qui suit, nous définissons ce sous-ensemble d'opérations dans les deux cas de permutation de deux opérations critiques et de replacement d'une opération critique sur une autre ressource potentielle.

Notons $JS(O_i)$ l'opération suivante de O_i selon la gamme opératoire du job de O_i. De même, notons $MS(O_i)$ l'opération suivante de O_i exécutée par la même ressource que O_i.

3.2.1 Permutation de deux opérations critiques

Soient O_i et O_j deux opérations critiques exécutées par la ressource R_k. Les seules opérations concernées éventuellement par une permutation de ces deux opérations sont les suivantes :

– $JS(O_i)$, $JS(JS(O_i))$, etc., $JS(O_j)$, $JS(JS(O_j))$, etc.

– $MS(O_i)$, $MS(MS(O_i))$, etc., $MS(O_j)$, $MS(MS(O_j))$, etc.

– $MS(JS(O_i))$, $MS(MS(JS(O_i)))$, etc., $MS(JS(O_j))$, $MS(MS(JS(O_j)))$, etc.

– $JS(MS(O_i))$, $JS(JS(MS(O_i)))$, etc., $JS(MS(O_j))$, $JS(JS(MS(O_j)))$, etc.

– *etc.*

3.2.2 Replacement d'une opération critique

Soit O_i une opération critique affectée à une ressource R_k et à replacer sur une ressource R_l à la date d. Soit O_x l'opération exécutée par R_l à l'instant d. Les opérations susceptibles à modifier sont les suivantes :

– $JS(O_i)$, $JS(JS(O_i))$, etc.

– $MS(O_i)$, $MS(MS(O_i))$, etc.

– $MS(JS(O_i))$, $MS(MS(JS(O_i)))$, etc.

– $JS(MS(O_i))$, $JS(JS(MS(O_i)))$, etc.

– O_x, $JS(O_x)$, $JS(JS(O_x))$, etc., $MS(JS(O_x))$, $MS(JS(JS(O_x)))$, etc.

– etc.

3.3 Solution initiale

Il a été démontré que l'efficacité des approches basées sur le principe de la recherche locale dépend étroitement de la qualité de la solution initiale choisie [10]. Dans notre approche, la solution initiale est déterminée par la collaboration d'une société d'agents. Dans la section suivante, nous présentons le modèle Multi-Agents ainsi que sa dynamique globale pour la détermination de la solution initiale et de la solution optimale basée sur la méthode de recherche tabou sus-mentionnée.

4 Modèle Multi-Agents

D'après la définition du problème Job Shop flexible, on dégage deux classes de contraintes : celles qui concernent les jobs, à savoir les contraintes de précédence et les contraintes temporelles, et celles relatives aux ressources, à savoir les contraintes disjonctives. Par conséquent, le modèle Multi-Agents proposé est constitué de deux classes d'agents : les *Agents Job* et les *agents Ressource* responsables de la satisfaction de ces deux types de contraintes. De plus, une troisième classe d'agents, contenant un seul agent, l'*agent Interface*, est ajoutée au modèle. Cet agent contient le noyau du processus de résolution, soit la méthode de recherche tabou. D'autre part, cet agent joue le rôle d'interface entre cette société d'agents et l'utilisateur.

Chaque agent composant notre modèle possède ses propres connaissances statiques et dynamiques et son propre comportement. Ce comportement dépend de son état : satisfait ou insatisfait et accorde une priorité au traitement des messages. De plus, chaque agent possède des accointances, càd les agents qu'il connaît et avec qui il peut communiquer. Une boîte aux lettres associée à chaque agent permet de sauvegarder les messages reçus des autres agents.

Dans la suite de cette section, nous détaillons les différents types d'agents.

4.1 Agent Job

L'agent Job peut communiquer avec tous les agents Ressource susceptibles d'exécuter ses opérations,

outre l'agent Interface. Ses connaissances statiques sont composées de sa date de début au plus tôt, sa date de fin au plus tard, sa gamme opératoire ainsi que les différentes durées de traitement de ses opérations. Ses connaissances dynamiques englobent les dates de début de ses différentes opérations ainsi que les ressources auxquelles elles sont affectées.

L'agent Job est satisfait lorsque toutes ses opérations ont été affectées à des ressources potentielles et toutes ses contraintes sont satisfaites, et dans ce cas il ne fait rien. Dans le cas contraire, il est insatisfait et il tente de placer une opération sur l'une des ressources susceptibles de l'exécuter en coopérant avec ses accointances.

4.2 Agent Ressource

L'agent Ressource peut communiquer avec tous les agents Job ayant des opérations susceptibles d'être exécutées par lui et de l'agent Interface. Ses connaissances statiques sont composées de la liste des opérations qu'il peut exécuter avec les durées respectives. Alors que ses connaissances dynamiques contiennent les opérations qui lui sont effectivement affectées avec les dates de début.

L'agent Ressource est satisfait lorsqu'il n'y a aucun conflit de chevauchement entre deux opérations qui lui sont affectées et dans ce cas il ne fait rien. Sinon, il est insatisfait et il essaye de résoudre les conflits en envoyant l'une des deux opérations en conflit à son agent Job pour qu'il la replace ailleurs.

4.3 Agent Interface

L'agent Interface possède comme accointances tous les agents du système. Ses connaissances statiques englobent :

- Le nombre d'itérations maximal permis

Ses connaissances dynamiques sont composées de :

- La liste tabou
- La solution courante ainsi que sont coût
- La meilleure solution rencontrée et son coût
- Le nombre d'itérations effectuées.

Comme on l'a déjà mentionné, le noyau de notre processus de résolution est implanté au niveau de l'agent Interface. Tant que le nombre d'itérations n'a pas atteint le nombre maximal, l'agent Interface est insatisfait. Sinon, il fournit la meilleure solution à l'utilisateur.

Dans le reste de ce papier, nous présentons la dynamique globale Multi-Agents pour les deux cas de détermination de la solution initiale et de la solution optimale.

5 Dynamique globale Multi-Agents

Dans cette section on s'intéresse à la description de la dynamique du système Multi-Agents proposé dans la résolution du problème Job Shop flexible. Celle-ci est décomposée essentiellement en deux phases : la détermination d'une solution initiale et l'application de la méthode de recherche tabou pour déterminer la solution optimale du problème.

5.1 Solution initiale

La solution initiale est le résultat de la coopération entre les différents agents du système. Initialement, l'agent Interface, connaissant le problème à résoudre, crée les différents agents Job et Ressource du système et demande aux agents Job de déterminer une affectation initiale pour chacune de leurs opérations via le message "*Déterminer_Affectation_Initiale(J_k)*". L'agent Job choisit alors une ressource parmi les ressources potentielles de l'opération et une date de début *d* de la façon suivante:

- S'il s'agit de la première opération du job (selon sa gamme opératoire) alors *d* est égale à la date de début au plus tôt du job.
- Sinon, *d* est égale à la date de fin de son opération précédente (*JP(O_i)*).

Pour le choix de la ressource, l'agent Job applique l'heuristique suivante : « la ressource la moins chargée parmi les ressources possibles est choisie pour l'opération courante ».

D'après cette affectation initiale, on remarque que les contraintes de précédence et les contraintes temporelles sont satisfaites, reste donc à satisfaire les contraintes disjonctives. A chaque fois qu'une opération a été affectée, l'agent Job informe l'agent Ressource concerné via le message "*Opération_placée (R_l, O_i, d)*". A la réception de ce message, l'agent Ressource R_l vérifie son état de satisfaction. Dans le cas où il est insatisfait, i.e. il y a un conflit de chevauchement entre cette opération et une autre opération qui lui est déjà affectée, il essaye de trouver un autre emplacement satisfaisant dont la date de début d_l soit la plus proche possible de *d*. S'il y en a un alors il informe l'agent Job via le message "*Opération_modifiée (J_k, O_i, d_l)*". Dans le cas contraire, l'agent Ressource éjecte l'opération et l'envoie à son agent Job pour qu'il lui trouve un autre emplacement via le message *"Opération_Refusée(J_k, O_i)"*. L'agent Job J_k envoie alors l'opération à une autre ressource potentielle via le message "*Placer_Opération(R_x, O_i)*".

Le processus sus-mentionné sera répété tant que l'opération n'a pas été affectée et pour un certain nombre d'itérations défini à l'avance. Une fois que cette limite est atteinte, c'est à dire une opération n'a pas trouvé un emplacement sur une des ressources

pouvant l'exécuter, l'agent Job demande à l'une des ressources de lui créer spécialement un emplacement via le message "*Créer_Emplacement* (R_x, O_i)". Cet emplacement doit satisfaire toutes les contraintes du problème. De même, si l'agent Ressource R_x échoue à créer un tel emplacement, il éjecte l'opération et la renvoie à son agent Job. La terminaison de cette première phase est assurée par une borne supérieure du nombre d'itérations pour la création d'une opération donnée (Pour plus de détails se référer à [7]) .

Afin de montrer l'efficacité de cette première phase à générer des solutions initiales de bonne qualité, nous avons effectué une série d'expérimentations sur des benchmarks (dimension d'ordre 15×15) et nous avons comparé les résultats obtenus avec l'approche de [11]. Les résultats sont illustrés par la figure 1.

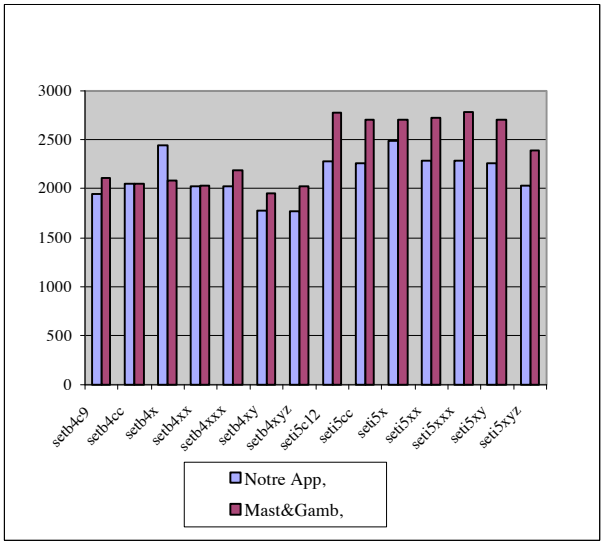

Figure 1. Résultats sur la solution initiale

5.2 Processus d'optimisation

Une fois que la première phase est terminée, l'agent Interface ayant reçu la solution initiale, il peut lancer le processus de recherche tabou. L'algorithme suivant présente le noyau de ce processus implanté au niveau de l'agent Interface.

```
1.  list_tabou ← ∅
2.  nb_iter ← 0
3.  sol_cour ← solution initiale
4.  meil_sol ← sol_cour
5.  Tant que nb_iter <= nb_iter_max faire
6.     iter ← 1
7.     Tant que iter <= iter_max & nb_iter <=
       nb_iter_max faire
8.        chemin ← chemin_critique (sol_cour)
9.        voisinage ← déterminer_voisinage (chemin)
10.       meil_vois←meilleur_voisin(voisinage)
11.       list_tabou←ajouter_dans_list_tabou
          (meil_vois)
12.       sol_cour ← effectuer_mvt (sol_cour,
          meil_vois)
13.       Si coût(sol_cour) < coût(meil_sol) alors
14.          meil_sol ← sol_cour
15.          nb_iter ← 0
          Fin si
16.          nb_iter ← nb_iter+1
17.          iter ← iter+1
18.       iter ← 0
       Fin tant que
19.    Diversification
    Fin tant que
```

Tableau 1. Algorithme Recherche Tabou utilisé

Une fois que le meilleur voisin dans le voisinage de la solution courante a été choisi, l'agent Interface envoie l'opération en question à son agent Job via le message "*Placer_opération* (J_k, O_i, R_l)" pour l'informer du mouvement à effectuer. A la réception de ce message, l'agent Job envoie à son tour cette opération à l'agent R_l pour qu'il place cette opération à une date de début d satisfaisant toutes les contraintes à travers le message "*Opération_placée*(R_l, O_i, d)". Le même processus décrit dans la première phase reprend.

Lorsque le nombre d'itérations effectuées après la dernière solution optimale dépasse un certain seuil "*iter_max*", une phase de diversification est lancée. Cette phase consiste à diversifier la recherche afin d'explorer de nouvelles zones de l'espace de recherche. Dans notre approche, une telle phase est caractérisée par le replacement d'un certain nombre d'opérations choisies aléatoirement. Le replacement d'une opération est effectué sur l'une de ses ressources potentielles choisie elle aussi de manière aléatoire.

6 Exemple d'illustration

Soit le problème job shop flexible suivant de dimension 3x3. Le tableau suivant illustre les durées de traitement des différentes opérations sur les différentes ressources.

	M_1	M_2	M_3
O_{11}	x	8	x
O_{12}	x	8	x
O_{13}	x	6	x
O_{21}	11	2	x
O_{22}	x	3	10
O_{23}	11	x	10
O_{31}	12	x	12
O_{32}	4	x	12
O_{33}	12	x	x

Tableau 2. Exemple de problème 3×3

La solution initiale élaborée par le système Multi-agents est illustrée par la figure 2. Le chemin critique est constitué des opérations O_{31}, O_{32}, O_{33}, O_{23}. Le coût de cette solution est 47.

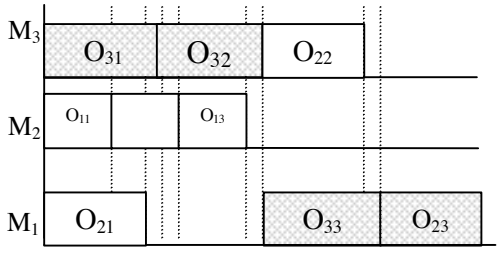

Figure 2. Solution initiale

Le voisinage de cette solution courante est composé des mouvements possibles suivants:

- Replacement de O_{31} sur M_1
- Replacement de O_{32} sur M_1
- Permutation de O_{33} et O_{23}
- Replacement de O_{23} sur M_3

Le meilleur voisin de ces derniers est le replacement de O_{32} sur M_1. En effet, le coût engendré est 39. La nouvelle solution est illustrée par la figure 3.

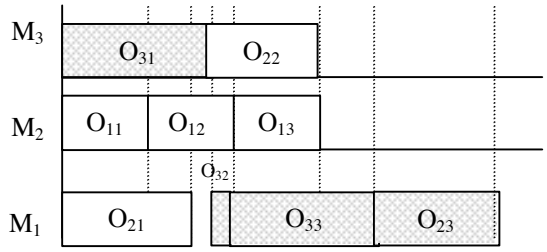

Figure 3. Itération 1 de la recherche tabou

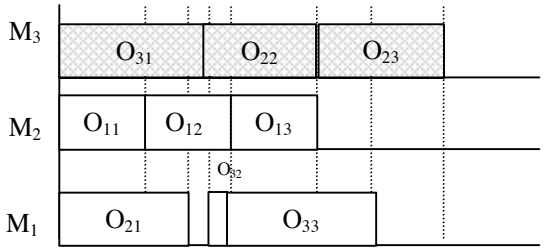

Figure 4. Itération 2 de la recherche tabou

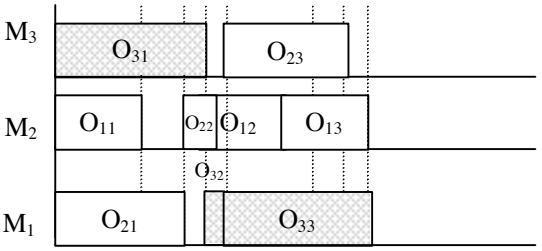

Figure 5. Solution optimale

La figure 5 montre la solution finale obtenue après le processus de résolution.

7 Expérimentation

Une série d'expérimentations a été effectuée sur les benchamrks définis par [1], [3], [4] et [9]. Ces benchmarks représentent des problèmes dont le nombre de jobs varie dans {10, 15, 20}, le nombre de ressources dans [5,20], le nombre d'opérations par job dans [5,25] et le nombre moyen de ressources potentielles par opération dans [1,3]. Ce qui donne des problèmes de nombre total d'opérations variant dans [50,500].

Pour chaque benchmark une suite d'exécutions a été effectuée à cause du nombre de paramètres de la recherche tabou à régler et aussi à cause du caractère aléatoire de la phase de diversification adoptée. Cinq exécutions ont été effectuées pour chaque instance et pour chaque paramètre. Les résultats illustrés dans ce qui suit présente le minimum trouvé après ces cinq exécutions.

Les paramètres de recherche tabou utilisés ont été fixés comme suit:

- Taille de la liste tabou variant dans {7,10,15,20,30}
- Nombre d'itérations total *nb_iter_max* fixé à 1000 itérations
- Nombre d'itérations entre deux phases de diversification *iter_max* variant dans {250,300,350}

Les figures 6 et 7 présentent une comparaison entre la solution initiale est la solution optimale fournies toutes les deux par notre approche ainsi que les bornes inférieures et supérieures trouvées dans la littérature pour ces instances. La figure 6 illustre les

benchmarks de [1] alors que la figure 7 illustre ceux de [9]. Ces deux figures montrent que notre approche fournit des solutions optimales appartenant à l'intervalle constitué par les bornes *inf* et *sup*.

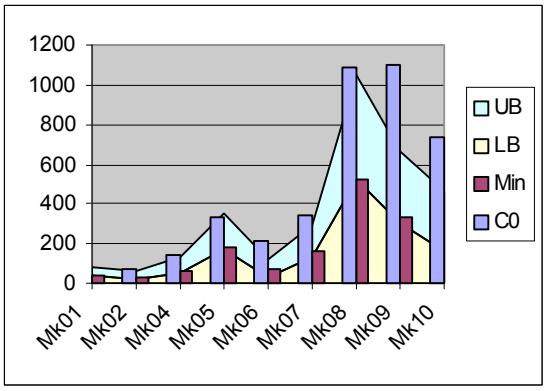

Figure 6. Benchmarks de Brandimarte

8 Conclusion

Dans le présent article, nous avons présenté une approche Multi-Agents pour la résolution du problème Job Shop flexible. Cette approche est basée sur la méthode de recherche tabou. Le système Multi-Agents est composé de trois classes d'agents: les agents Job, les agents Ressource et un agent Interface. Chaque classe d'agents est responsable de la satisfaction des contraintes qui lui sont relatives. Une série d'expérimentation a été effectuée sur des benchmarks connus. Les résultats présentés montrent un net écart entre la solution initiale et la solution optimale obtenue après le processus de recherche tabou ainsi que l'appartenance de la solution trouvée à l'intervalle formé par les bornes inférieures et supérieures définies dans la littérature. Les travaux actuels portent sur la distribution de la méthode de recherche tabou au niveau des différents agents de telle sorte que la décision actuellement prise d'une façon globale au niveau de l'agent Interface sera répartie entre les différents agents. En plus, une comparaison avec une approche centralisée est en train d'être effectuée.

Bibliographie

[1] Brandimarte P., *Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 22*, pp 158-183, 1993.

[2] Brucker, P. et Neyer, J., *Tabu-search for the multi-mode job-shop problem* , OR Spektrum 20, 21-28, 1998.

[3] Chambers et Barnes, *Flexible Job Shop scheduling by tabu search*, Graduate program in Operations Research and Industrial Engineering, The university of Texas at Austin, Technical Report series, ORP96-09, 1996.

[4] Dauzerre-Peres S., Paulli J., *An integrated approach for modeling and solving the general multi-processor job shop scheduling problem using tabu search, Annals of Operations Research 70*, 281-306, 1997.

[5] Eikelder T., H.M.M., Aarts, B. J. M., Verhoeven, M. G. A. and Aarts, E.H.L., *Sequential and Parallel Local Search Algorithms for Job Shop Scheduling, MIC'97 Proceedings of the 2nd International Conference on Meta-heuristics*, Sophia-Antipolis, France, 21-24 July, pp. 75-80, 1997.

[6] Garey R. et Johnson D. S., *Computers and Intractability: A guide to the theory of NP-completeness,* Freeman and Co, 1979.

[7] Ghédira K. et Ennigrou M., *How to schedule o Job Shop Problem through agent cooperation, AIMSA 2000: Aritificial Intelligence: Methodology, Systems, Architectures*, 2000.

[8] Glover F., *Future paths for Integer Programming and Links to Artificial Intelligence*, *Computers and Operations Research*, 5:533-549, 1986.

[9] Hurink E., Jurisch B., Thole M., *Tabu search for the Job Shop scheduling problem with multi-purpose machine, Operations Research Spektrum 15*, 205-215, 1994.

[10] Jain A., Rangaswamy B., Meeran S., *Job shop neighbourhoods and Move evaluation strategies*, 2000.

[11] Mastrolilli M., Gambardella L.M., *Effective Neighborhood Functions for the Flexible Job Shop Problem, Journal of Scheduling*, Volume 3, Issue 1, 2000. Pages:3-20, 2000.

[12] Nuijten W., Aarts E., *A computational study of Constraint Satisfaction for multiple capacitated Job Shop scheduling, European Journal of Operations Research,* 90(2): 269-284, 1996.

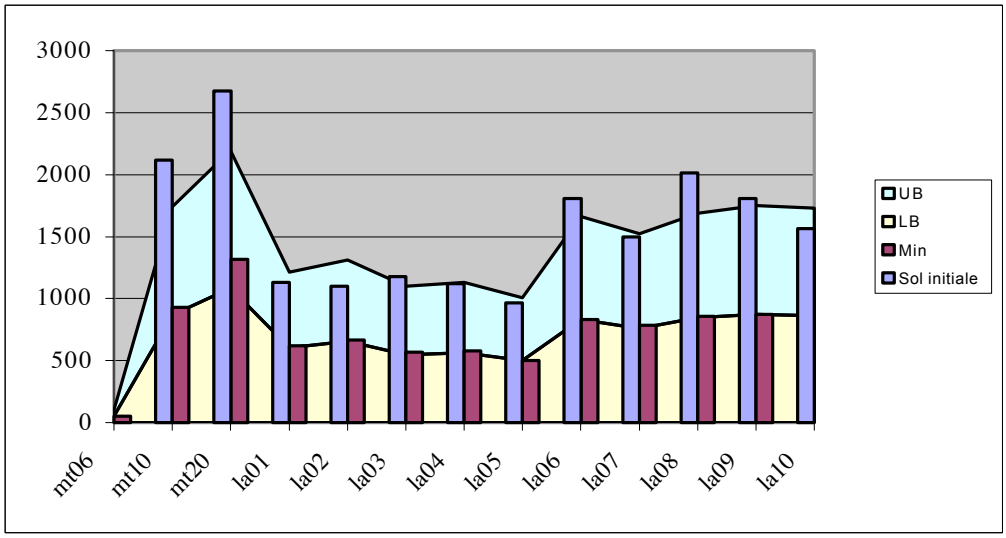

Figure 7. Benchmarks de Hurink et al.