Les Annales RECITS (http://www.lrecits.usthb.dz)
Vol. 02, 2015, pages 27-39

Métaheuristiques pour un problème d'ordonnancement sous contraintes de préparation

Wafaa LABBI et Mourad BOUDHAR

Laboratoire RECITS, Faculté de Mathématiques, USTHB
BP 32 El-Alia, BEZ, Alger, Algerie

fawalab@yahoo.fr, mboudhar@yahoo.fr

Abstract: In this paper, we study the scheduling problem under preparation time constraints. This consists in scheduling a set of independent jobs on a set of identical parallel machines. We assume that there are k types of renewable resources needed for the preparation of the jobs. Each job has an execution time and requires prior to its execution a preparation time completed by a subset of resources. The objective is to find a schedule that minimizes the makespan. We have proposed metaheuristics to solve this problem and conducted with numerical experiments.

Keywords: Scheduling, parallel machines, makespan, heuristics, metaheuristics, resources, preparation times.

Résumé : Nous considérons le problème d'ordonnancement sous contraintes de préparation. Il consiste à ordonnancer un ensemble de tâches indépendantes sur un ensemble de machines parallèles identiques. Nous supposons qu'il existe k types de ressources renouvelables nécessaires à la préparation des tâches. Chaque tâche possède pour son traitement un temps d'exécution qui doit être précédée par une phase de préparation réalisée par un sous ensemble de ressources. L'objectif est de minimiser le makespen. Nous avons proposé des métaheuristiques pour résoudre ce dernier suivies d'une étude expérimentale.

Mots clés : Ordonnancement, machines parallèles, makespan, heuristiques, métaheuristiques, ressources, temps de préparation.

1 Introduction

Le problème que nous abordons dans ce papier est un problème d'ordonnancement sur machines identiques dont le but est de minimiser la durée totale (C_{max}). Nous nous intéressons particulièrement au cas où une tâche nécessite avant son exécution un temps de préparation réalisé par un sous ensemble de ressources renouvelables. Ce n'est qu'à partir des années 1960 que les chercheurs ont commencé à s'intéresser à ce type de problèmes. Les principaux résultats ont été résumés dans [4, 22, 5].

Le problème d'ordonnancement sous contraintes de préparation peut être décrit comme suit : Un ensemble $T = \{T_1, T_2, ..., T_n\}$ de n tâches indépendantes doit être exécuté sur un ensemble de m machines parallèles identiques. Outre que les machines, nous supposons qu'il existe un ensemble de k types de ressources renouvelables $R = \{R_1, R_2, ..., R_k\}$ disponible pour la phase de préparation où chacun est disponible en une unité. Le traitement de chaque tâche T_i nécessite un temps de préparation s_i, un temps d'exécution p_i et un sous-ensemble de ressources. Durant la phase de préparation, la machine n'est pas disponible pour une autre tâche et l'exécution d'une tâche doit être effectuée immédiatement après sa préparation. L'interruption de la préparation et de l'exécution des tâches n'est pas autorisée.

Ce problème peut trouver plusieurs applications en industrie, par exemple, il peut être issu du processus de fabrication de pneumatique. La fabrication d'un pneu passe par plusieurs phases et chaque phase du processus de fabrication requérant une grande précision, et d'importants contrôles. Nous nous intéressons de prés dans ce travail à l'étape de cuisson qui nécessite un ensemble de ressources (tel que la main-d'oeuvre, les outils) qualifié pour commencer la période de préparation de la charge de moule et le positionnement des pneus sur les machines, le pneu est au curry après la phase de préparation. Dans cette situation, le temps de traitement d'une tâche sur une machine est ainsi décomposé en deux parties : le temps de préparation et le temps d'exécution. La période de préparation doit précéder la période d'exécution pour chaque tâche et exige un certain type de ressources. Ce type de problème peut être rencontré aussi en informatique dans un environnement multiprocesseurs ou dans un réseau d'ordinateurs en présence de serveurs. Le serveur qui est une ressource se charge de l'allocation des processus aux ordinateurs ce qui correspond à la phase de préparation et le traitement des processus sur les processeurs correspond à la phase d'exécution.

Les problèmes d'ordonnancement sous contraintes de ressources pour l'affectation des tâches ont reçu peu d'attention dans la littérature, Abdelkhodaee et Wirth [1] ont étudié le problème d'ordonnancement sur deux machines parallèles identiques avec ressource supplémentaire qui est un serveur. Chaque tâche nécessite deux opérations, la première est l'opération de setup traitée par le serveur, alors que la deuxième opération est exécutée automatiquement par l'une des machines parallèles (sans serveur). L'objectif est de minimiser le makespan. Les auteurs ont prouvé que ce problème est NP-difficile au sens fort. Ils ont proposé une formulation mathématique en nombres entiers, ont présenté deux sous problèmes polynomiaux, et ont aussi proposé deux heuristiques avec des expérimentations numériques pour le problème général. L'opération de setup correspond à la phase de préparation dans notre problème, les auteurs ont relaxé la contrainte que la phase d'exécution

d'une tâche doit être effectuée immédiatement après la phase de préparation. Abdekhodaee et al. [2] ont discuté deux cas particuliers où les temps d'exécution sont identiques et les temps de setup sont aussi identiques. Ils ont montré que ces deux problème sont NP-difficile au sens faible et ont proposé des bornes inférieures ainsi que des heuristiques constructives. Suite aux travaux cités en [3], Gan et al. [14] ont développé deux formulations de programmation linéaire mixte en nombres entiers et deux variantes of a branch-and-price scheme. Koulamas [18] a traité le problème d'ordonnancement à deux machines parallèles Semi-automatiques pour minimiser le temps mort résultant de l'indisponibilité du robot avec la condition que ces deux machines partagent le même serveur (robot) pour la préparation des tâches, il a démontré que ce dernier est NP-difficile au sens fort, aussi il a développé une procédure de réduction pour le transformer en un problème de taille plus petite. Kravchenko et Werner [20] ont considéré le problème d'ordonnancement de m machines parallèles en présence d'un seul serveur dont l'objectif est de minimiser le makespan, ils ont présenté un algorithme pseudo-polynomial pour le cas de deux machines où les temps de préparation sont unitaires et ont prouvé que le problème avec un nombre arbitraire de machines et les temps de préparation sont égaux à 1 est NP-difficile, ils ont proposé des heuristiques pour sa résolution. Dans [23] les mêmes auteurs ont étudié le problème d'ordonnancement où avant le début de l'exécution d'une tâche, un temps de setup est nécessaire et doit être effectué par un ensemble de serveurs. Ils ont proposé un algorithme pseudo-polynomial pour le problème avec des temps de setup unitaires, m machines et $m-1$ serveurs et ont montré que le problème avec un nombre fixe de machines et de serveurs pour minimiser le retard maximum est NP-difficile au sens faible. Aussi, ils ont généralisé des algorithmes pour les problèmes d'ordonnancement à machines parallèles avec des temps de traitement constants au problème correspondant avec serveurs lorsque $s_i = s$ et ils ont donné l'analyse du plus mauvais cas des deux heuristiques de liste. Hasani et al. [17] ont considéré le problème d'ordonnancement sur deux machines identiques avec un seul serveur pour minimiser le makespan. Avant le traitement, chaque tâche doit être chargée sur une machine et prend un temps donné de setup qui doit être effectué par le serveur. Ils ont proposé une formulation de programmation linéaire mixte en nombres entiers en utilisant une simple idée qui consiste à une décomposition possible de l'ordonnancement en un ensemble de blocs et ils ont comparé la performance de ce modèle avec des heuristiques proposées dans [14]. Les expérimentations numériques ont montré que ce modèle performe mieux que les heuristiques.

Lorsque les ressources sont nécessaires durant l'exécution des tâches, plusieures recherches sont considérées dans la littérature, en particulier pour problèmes d'ordonnancement avec machines parallèles, [15, 9, 7]. Garey and Johnson [15] ont montré que le problème $P2|res\cdot\cdot\cdot, p_i = p|C_{max}$ est résolu polynomialement en $O(n^{5/2})$, alors que le problème $P2|res1\cdot\cdot, p_i = 1|C_{max}$ est résolu en $O(nlogn)$. Blazewicz et Ecker [10] ont montré que le problème $P|res\, sor, p_i = 1|C_{max}$ (où le nombre de types de ressources, les limites de ressources et les besoins en ressources sont fixés par les entiers positifs s, o, r, respectivement) est résolu en $O(n)$, même pour un nombre arbitraire de ressources, où les temps de traitement des tâches appartiennent à un nombre fixe de classes L, ou les tâches de même classe ont le même traitement et les besoins en ressources. Blazewicz et al. [11] ont montré que le problème $Pm|res\, sor|C_{max}$ est résolu en $O(n^{L(m+1)})$, lorsque le nombre de machines m est fixé. Blazewicz et al. [8] ont montré que le problème $P2|res\cdot 11, p_i = 1, r_j|C_{max}$ est NP-difficile au sens fort et que ce dernier est un cas particulier du problème $P2, |res^t\cdot 11, s_i = 1, p_i = p, r_i|C_{max}$ en posant $p = 0$ et en remplaçant $s_i = 1$ par $p_i = 1$, donc le problème

$P2, |res^t \cdot 11, s_i = 1, p_i = p, r_i|C_{max}$ est NP-difficile au sens fort. Dans la section 2, nous présentons les approches métaheuristques proposées pour résoudre le problème général. La section 3 porte sur l'évaluation numériques de la performance des méthodes proposés et une conclusion clôt ce tapuscrit.

2 Approche de résolution

Les métaheuristiques sont des heuristiques stochastiques itératives qui progressent vers un optimum local en se comportant comme des algorithmes de recherche, ces méthodes sont généralement hybridées avec une recherche locale. Elles sont souvent inspirées par des analogies avec des phénomènes de la nature, la physique (Recuit simulé), la biologie (algorithmes évolutionnaires) et l'éthologie (colonies de fourmis). Parmi ces méthodes, nous allons adopter deux d'entre elles : le recuit simulé et l'algorithme génétique.

2.1 Recuit simulé

Le recuit simulé est une métaheuristique inspirée d'un processus utilisé en métallurgie. Ce processus alterne des cycles de refroidissement lent et de réchauffage (recuit) qui tendent à minimiser l'énergie du matériau. Elle est aujourd'hui utilisée en optimisation pour trouver les extrémaux d'une fonction. Elle a été mise au point par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983 [19], et indépendamment par V. Cerny en 1985 [12]. Le recuit simulé s'appuie sur l'algorithme de Metropolis, qui permet de décrire le comportement d'un système en équilibre thermodynamique à une certaine température T, partant d'une configuration donnée (solution initiale). Par analogie avec le processus physique, la fonction objectif à minimiser deviendra l'énergie E du système. Les étapes du recuit simulé adaptées à notre problème sont les suivantes :

– Solution initiale : La solution initiale que nous avons considéré est représentée par la liste de priorité de la meilleure heuristique (voir [21]).
– Voisinage : Le mécanisme proposé pour générer des voisins d'une solution est le suivant : sans perte de généralité, supposons que le $C_{max} = maximum(C_j)$ $(j = 1..., m)$, est sur la machine M_1 et le $minimum(C_j)$ est sur la machine M_2, nous calculons l'écart $\Delta C = C_1 - C_2$. S'il existe une tâche T_i qui est traitée sur la machine M_1 avec un temps de traitement inférieur à ΔC, on la déplace à la dernière position sur la machine M_2 si elle respecte les contraintes de ressources. Sinon, on permute entre deux tâches quelconques dans la liste de priorité décrivant la solution en cours, on note le voisinage d'une liste l l'ensemble des listes $N(l)$.

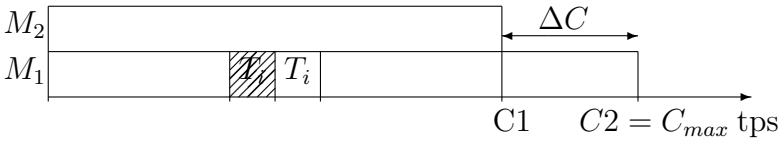

FIGURE 1 – Voisinage.

– Fonction d'évaluation : L'objectif du notre problème est de minimiser la date de fin de traitement de l'ensemble des tâches qui correspond à la minimisation de l'énergie du système. A toute solution du problème, nous appliquons les sélections Sérielle et Parallèle et nous choisissons la meilleure des deux.

Le schéma général de la méthode est donné par l'algorithme suivant :

Algorithme de recuit simulé pour le problème $Pm|res^t \cdot 11|C_{max}$

1. Initialiser l la liste de départ.
2. Initialiser une température maximale T et une température minimale $Tmin$.
3. **Tantque** $(T > Tmin)$ **faire**
4. - Choisir un voisin $l' \in N(l)$
5. - Calculer ΔC : $\Delta C = C(l') - C(l)$
6. **Si** $\Delta C < 0$ **then** $l' = l$
7. **Sinon**
8. - Générer aléatoirement une probabilité r
9. **Si** $r \leq \exp \frac{-\Delta C}{T}$ **then** $l' = l$
10. **Finsi**
11. **Finsi**
12. $T = T * (1 - alpha)$
13. **Fin**

2.2 Algorithme génétique

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires. Ils s'inspirent de l'évolution biologique des espèces et basées sur une imitation des phénomènes d'adaptation des êtres vivants. Avec ce type de méthodes, il ne s'agit pas de trouver une solution analytique exacte mais de trouver une bonne solution satisfaisante dans un temps de calcul raisonnable. La première description du processus des algorithmes génétiques a été donnée par Holland en 1975 [13], puis Goldberg [16] les a utilisés pour résoudre des problèmes concrets d'optimisation.

Le but de ces algorithmes génétiques est d'optimiser une fonction prédéfinie, appelée fonction objectif, ou fitness ; ils travaillent sur un ensemble de solutions candidates, appelé *population* d'individus ou chromosomes. Ces derniers sont constitués d'un ensemble d'éléments, appelés *gènes*, qui peuvent prendre plusieurs valeurs, appelées *allèles*. Un chromosome est une représentation ou un codage d'une solution du problème donné. Une première population est choisie soit aléatoirement, soit par des heuristiques ou par des méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques. Cette population doit être suffisamment diversifiée pour que l'algorithme ne reste pas bloqué dans un optimum local. C'est ce qui se produit lorsque trop d'individus sont semblables. Les algorithmes génétiques génèrent de nouveaux individus, de telle sorte qu'ils soient plus performants que leurs prédécesseurs. Le processus d'amélioration des individus s'effectue par utilisation d'opérateurs génétiques, qui sont la sélection, le croisement et la mutation. Les étapes de l'algorithme génétique adaptées à notre problème sont les suivantes :

- Codage : Le codage que nous avons retenu est de longueur égale au nombre total de tâches à réaliser. L'individu est alors représenté par une séquence de tâches. La Figure **??** présente le codage d'une solution quelconque. La tâche numéro 5 sera lancé en production la première, suivi de la tâche numéro 7... etc. et enfin la tâche numéro 6, placé dans le dernier chromosome du code, sera lancée la dernière.

- Population initiale : la population initiale que nous avons considéré est composée de séquences de tâches utilisant les règles de priorités définies dans [21] et de séquences de tâches générées aléatoirement. Le problème principal dans cette étape est le choix de la taille de la population. Si la taille de la population est trop grande, le temps de calcul augmente et demande un espace mémoire important. Par contre, une population de taille très petite, la solution obtenue n'est pas satisfaisante. Il faut donc trouver le bon compromis. Pour cela, nous avons testé le paramètre T_{int} qui représente la taille de notre population avec les valeurs $\{10, 20, 50, 100, 200\}$ et d'après les résultats obtenus, la bonne valeur est $T_{int} = 50$.
- La procédure de sélection : la sélection permet d'identifier les individus susceptibles d'être croisés dans une population. La procédure de sélection que nous avons utilisée est la sélection aléatoire, cette sélection se fait aléatoirement, uniformément et sans intervention de la valeur de la fonction objectif, donc chaque individu a une probabilité uniforme $\frac{1}{T_{int}}$ d'être sélectionné.
- Le croisement : le croisement permet d'enrichir la population en manipulant les composantes des chromosomes. Un croisement est envisagé avec deux parents et génère un ou deux enfants. Les individus survivants à la phase de sélection vont subir le croisement avec une probabilité P_{crois}, dans notre algorithme génétique, nous utilisons le croisement en 2-points. Ce dernier s'effectuera de la manière suivante :

  - Choisir deux individus de la population actuelle comme parents pour générer deux enfants.
  - Générer aléatoirement deux positions $k_1, k_2 \in [1, ..., n]$ avec $k_1 < k_2$.
  - Générer aléatoirement une probabilité de croisement α.
  - Si $\alpha \leq P_{crois}$: ce croisement conserve dans l'enfant i la zone interne du parent j (zone comprise entre k_1 et k_2)et pour l'enfant j la zone interne du parent i. Ensuite, compléter les cases vides restantes de l'enfant i par les éléments du parent i et les cases vides restantes de l'enfant j par les éléments du parent j qui ne provoquent pas de doublon.
  - Si $\alpha > P_{crois}$: copier le parent i à l'enfant i et le parent j à l'enfant j.
- La mutation : les individus issus du croisement vont ensuite subir un processus de mutation avec une probabilité P_{mut}. La mutation nous garantit que l'algorithme génétique sera susceptible d'atteindre la plupart des points du domaine réalisable. L'opérateur de mutation utilisé dans notre cas est l'opérateur d'échange réciproque qui permet de sélectionner au hasard deux gènes et de les échanger. Si β, généré aléatoirement, appartient à l'intervalle $[0, P_{mut}]$ nous appliquons l'opérateur de mutation sur l'enfant i pour avoir un enfant muté $imut$, et si $\beta > P_{mut}$ nous appliquons l'opérateur de mutation sur l'enfant j pour avoir un enfant muté $jmut$.
- Fonction d'évaluation : pour chaque chromosome, qui est une permutation de n tâches, nous appliquons les sélections Sérielle et Parallèle, ensuite nous choisissons la meilleure des deux solutions trouvées.
- Insertion : les individus qui sont générés aléatoirement et les enfants mutés sont triés

selon leur fonction d'évaluation dans un ordre croissant. Seule la moitié supérieure de la population, correspondant aux meilleurs individus, est sélectionnée. Il est à noter que la taille de la population reste fixe égale à T_{int} de génération en génération.
– Critère d'arrêt : l'algorithme génétique s'arrête après un nombre fixé de génération noté $Itermax$.

Finalement, les étapes de notre algorithme sont données par l'algorithme suivant :

Algorithm génétique pour le problème $Pm|res^t \cdot 11|C_{max}$

1. Initialiser : T_{int}, $Itermax$, P_{mut}, P_{croi}, δ.
2. Générer la population initiale de taille T_{int}.
3. **Répéter**
4. i=0
5. **Tant que** $\delta < \frac{T_{int}}{2}$ **faire**
6. Sélectionner aléatoirement deux parents de la population.
7. Croisement des deux parents pour obtenir deux enfants par une probabilité P_{croi}.
8. Muter les deux enfants par une probabilité P_{mut} pour obtenir un enfant muté.
9. $\delta = \delta + 1$.
10. **Fin Tant que.**
11. Evaluer tous les chromosomes par la fonction d'évaluation.
12. Ranger les parents et les enfants dans l'ordre croissant selon leur fonction d'évaluation.
13. Supprimer les T_{int} chromosomes faibles et enregistrer les T_{int} meilleures chromosomes selon leur fonction d'évaluation.
14. i=i+1.
15. **Jusqu'à** $i = Itermax$.

2.3 Algorithme génétique hybride

Les algorithmes génétiques possèdent une connaissance globale à travers leurs populations et explorent l'espace de recherche tandis que le recuit simulé détient une connaissance locale et exploite le voisinage. Dans cette partie, nous proposons un algorithme génétique hybride séquentielle. Notre approche se fait en trois étapes.

1. La première est une application directe de l'algorithme génétique et à la fin de cette dernière, une population d'individus qui satisfait notre critère est obtenue.
2. Dans une deuxième étape, on a recours au recuit simulé. Dans ce cas, le meilleur individu de la population finale obtenu dans la première étape est choisi comme solution initiale.
3. A la fin de la deuxième étape, on obtient une solution qui va être améliorée dans la troisième étape par l'application de l'un des algorithmes de liste basés sur les rangements des tâches définis ci-dessous.
   – CJ : A l'étape i, si la tâche $T_{j'}$ qui est à la position $i+1$ est compatible avec la tâche T_j qui est à la position i on la laisse à cette position, sinon on cherche une tâche compatible avec cette dernière et on la permute avec la tâche $T_{j'}$.

– CJI : On prend l'inverse du rangement CJ.
– ICJ : A l'étape i, si la tâche $T_{j'}$ qui est à la position $i+1$ est incompatible avec la tâche T_j qui est à la position i on la laisse à cette position, sinon on cherche une tâche incompatible avec cette dernière et on la permute avec la tâche $J_{j'}$.
– $ICJI$: On prend l'inverse du rangement ICJ.
– ER : On sélectionne deux tâches au hasard et on les permute.

Pour le choix des meilleures listes, nous avons mené une étude comparative entre eux en leurs appliquant les sélections Sérielle et Parallèle. Nous avons généré des instances aléatoires du problème pour les valeurs de m et n appartenant aux ensembles $\{2, 5, 10\}$, $\{50, 100, 500, 1000\}$, respectivement. Pour chaque valeur de n, nous avons testé 100 instances. Les temps de préparation et d'exécution des tâches sont tirés suivant la loi uniforme dans les intervalles [1, 10], [10, 50] et [50, 100]. Nous avons comparé ces heuristiques les unes par rapport aux autres en calculant le nombre de fois où une heuristique fournit de meilleures solutions. Les résultats des expérimentations numériques sont donnés dans la Table 1.

n	$\|R\|$	$s_i, p_i \in [1,10]$					$s_i, p_i \in [10,50]$					$s_i, p_i \in [50,100]$				
		CJ	**CJI**	**ICJ**	**ICJI**	**ER**	**CJ**	**CJI**	**ICJ**	**ICJI**	**ER**	**CJ**	**CJI**	**ICJ**	**ICJI**	**ER**
m=2																
50	1	12	59	34	0	20	13	46	30	0	17	7	53	37	0	4
	2	3	50	52	0	19	5	48	43	0	10	4	39	45	0	12
	5	2	58	24	11	18	4	58	17	7	14	15	44	20	10	12
	7	12	61	13	10	16	10	56	9	17	12	18	33	17	18	16
100	1	14	50	46	0	22	9	46	39	0	15	8	46	42	0	8
	2	1	49	51	0	9	2	54	40	0	9	2	56	40	0	5
	5	0	89	10	0	5	0	81	15	0	5	7	60	20	2	12
	7	0	88	7	5	5	3	72	15	7	5	22	40	14	12	13
500	1	2	50	52	0	24	4	41	42	0	18	2	47	46	0	6
	2	1	47	46	0	22	0	50	47	0	9	0	55	43	0	2
	5	0	99	4	0	0	0	97	3	0	0	0	94	6	0	0
	7	0	100	0	0	0	0	99	1	0	0	0	96	3	0	1
1000	1	7	46	47	0	28	3	47	40	0	15	1	38	53	0	8
	2	0	56	46	0	10	0	50	45	0	7	0	50	49	0	2
	5	0	95	9	0	0	0	99	1	0	0	0	99	1	0	0
	7	0	100	0	0	0	0	100	0	0	0	0	100	0	0	0
m=5																
50	1	21	44	34	0	17	28	31	20	0	21	23	32	29	0	16
	2	19	43	28	2	26	23	39	21	1	24	14	52	21	2	14
	5	4	68	17	13	12	7	62	16	7	15	4	71	15	9	8
	7	3	57	25	15	22	9	57	17	16	11	4	64	16	10	13
100	1	36	25	21	0	22	31	29	21	0	22	18	28	27	0	28
	2	16	52	20	0	22	18	45	21	0	19	7	65	14	0	15
	5	0	80	11	4	9	2	80	9	3	6	0	78	10	8	5
	7	2	85	8	4	5	1	82	4	5	10	1	86	7	4	5
500	1	20	31	22	0	30	17	37	36	0	11	25	26	25	0	25
	2	14	58	20	0	17	8	67	12	0	13	11	66	14	0	9
	5	0	97	2	0	1	0	99	1	0	0	0	98	0	0	2
	7	0	100	0	0	0	0	99	0	0	1	0	100	0	0	0
1000	1	19	41	24	0	20	21	31	26	0	22	22	25	27	0	26
	2	7	61	18	0	18	3	66	14	0	17	7	65	14	0	14
	5	0	98	2	0	0	0	98	1	0	1	0	99	1	0	0
	7	0	99	1	0	0	0	100	0	0	0	0	100	0	0	0
m=10																
50	1	38	45	29	0	30	27	36	23	0	21	29	40	24	0	23
	2	27	48	35	7	35	27	39	34	5	25	30	48	29	6	28
	5	7	51	25	21	19	10	42	25	20	20	6	63	20	6	13
	7	13	56	23	19	23	5	63	16	11	18	8	55	18	15	13
100	1	27	41	25	0	23	27	33	16	0	28	27	31	28	0	17
	2	23	45	30	0	22	21	38	21	0	25	20	39	21	2	30
	5	3	65	18	11	13	5	58	18	11	13	4	67	9	11	12
	7	1	78	11	10	10	1	78	9	9	9	0	78	10	6	8
500	1	33	30	25	0	28	28	27	24	0	23	18	22	27	0	33
	2	16	54	21	0	23	19	42	22	0	17	16	43	22	0	19
	5	0	90	8	1	1	0	76	11	4	10	3	66	18	4	9
	7	0	99	0	0	1	0	97	1	0	2	0	95	1	0	4
1000	1	29	27	20	0	31	27	24	24	0	27	21	32	25	0	22
	2	20	50	18	0	19	17	46	26	0	17	14	47	19	0	20
	5	0	96	3	0	1	0	71	14	0	15	3	62	15	5	15
	7	0	100	0	0	0	0	96	1	0	3	0	96	3	0	1

TABLE 1 – Comparaison entre les algorithmes de liste.

A partir de la Table 1, nous constatons que l'heuristique basée sur la liste CJI donne de meilleurs résultats pour toutes les instances testées. Nous remarquons qu'elle est meilleure à 100% pour la plupart des instances à 500 et 1000 tâches avec 5 et 7 types de ressources. L'heuristique basée sur la liste ICJ fournit des résultats satisfaisants pour les instances à 50 et 100 tâches avec 1 et 2 types de ressources.

3 Expérimentations numériques

Dans la section précédente, nous avons discuté des approches métaheuristiques que nous avons adapté pour la résolution du problème $Pm|res^t \cdot 11|C_{max}$. Les métaheuristiques disposent de mécanismes particuliers permettant d'éviter d'être rapidement piégées par les minimas locaux. Ces méthodes se montrent donc le plus souvent plus puissantes que les heuristiques. Cependant, le bon réglage des différents paramètres est l'étape cruciale pour que la métaheuristique retourne de bons résultats. Pour le recuit simulé, nous avons fixé ses paramètres de la manière suivante : Température initiale $T = 100$ (nous avons testé le paramètre T avec plusieurs valeurs 10, 50, 100 et 500, et d'après les résultats obtenus, la bonne valeur est $T = 100$ car nous n'avons remarqué aucune amélioration de la solution pour les autres valeurs). Température minimale $T_{min} = 10^{-4}$, donc l'algorithme s'arrête quand $T < 10^{-4}$ et pour diminuer la température nous avons choisi $\alpha = 0,3$. Le nombre de paliers effectués vaut donc 40. Pour les paramètres de l'algorithme génétique et après plusieurs expérimentations nous avons pu les fixer comme suit : Une taille de la population égale à 50, une probabilité de croisement $P_{crois} = 0.8$, une probabilité de mutation $P_{mut} = 0.1$ et en fin l'algorithme s'arrête après 100 itérations.

L'objectif de cette section est de comparer les performances de l'algorithme génétique (AG), l'algorithme génétique hybride (AGH) et le recuit simulé (RS). Nous avons généré des problèmes tests possédant différentes caractéristiques, dans un premier temps nous avons fixé le nombre de machines à $m = 2$ et 5, et pour chaque nombre de machines nous avons fait varier le nombre de tâches ($n = 50, 100, 500$) et pour chaque nombre de tâches nous avons fait varier le nombre de types de ressources ($k = 1, 2, 5, 7$). Les temps de préparation et d'exécution sont générés selon une loi uniforme dans les intervalles suivants : [1, 10],[10, 50] et [50, 100]. Les résultats de ces méthodes ont été comparés avec la borne inférieure LB. Nous avons calculé le nombre de fois où une métaheuristique fournit de meilleures solutions par rapport aux autres et le nombre de fois où la solution trouvée par une métaheuristique est égale à la borne inférieure. Aussi, nous avons calculé les déviations moyenne et maximale. Les Tables 2 et 3 illustrent ces résultats.

Sur l'ensemble des tests effectués, l'algorithme génétique se montre efficace dans la résolution de la majorité des problèmes. Il est capable de résoudre à l'optimalité tous les problèmes à 50, 100 et 500 tâches avec 1, 2 et 5 types de ressources quand les temps de préparation et d'exécution prennent leurs valeurs dans l'intervalle [1, 10]. Nous remarquons aussi que pour les instances à $n = 100$ et 500 tâches, l'algorithme génétique et l'algorithme génétique hybride sont proches en terme de performance. Toutefois, l'algorithme génétique hybride devient meilleur pour les instances à 5 et 7 types de ressources quand les temps de préparation et d'exécution prennent leurs valeurs dans les intervalles [10, 50] et [50, 100]. Par ailleurs, le recuit simulé est le moins performant des trois.

n	$\|R\|$		$s_i, p_i \in [1,10]$			$s_i, p_i \in [10,50]$			$s_i, p_i \in [50,100]$		
			AG	**AGH**	**RS**	**AG**	**AGH**	**RS**	**AG**	**AGH**	**RS**
50	1	#Best	100	86	79	100	79	14	98	92	5
		#LB	100	86	79	100	79	14	95	90	5
		$\%Dev_{max}$	0	1,06	6,36	0	0,59	5,31	0,4	0,32	4,0
		$\%Dev_{moy}$	0	0,06	0,29	0	0,03	0,77	0,008	0,01	1,28
	2	#Best	100	74	54	96	88	4	94	83	1
		#LB	100	74	54	96	88	4	88	78	1
		$\%Dev_{max}$	0	4,16	14,23	0,13	0,32	7,58	0,71	0,6	4,39
		$\%Dev_{moy}$	0	0,21	0,93	0,003	0,01	1,56	0,02	0,03	1,82
	5	#Best	97	53	2	81	71	0	61	72	0
		#LB	95	52	2	50	51	0	36	37	0
		$\%Dev_{max}$	0,37	5,0	15,38	1,67	2,16	9,58	0,9	1,05	8,32
		$\%Dev_{moy}$	0,01	0,74	4,33	0,14	0,21	4,41	0,17	0,14	3,47
	7	#Best	90	62	0	54	59	0	62	44	0
		#LB	52	38	0	13	9	0	1	2	0
		$\%Dev_{max}$	2,43	10,76	18,05	2,15	1,91	12,05	1,63	1,45	8,73
		$\%Dev_{moy}$	0,28	1,07	6,44	0,52	0,49	6,57	0,5	0,56	5,21
100	1	#Best	100	84	84	100	74	16	90	97	1
		#LB	100	84	84	100	74	16	90	97	1
		$\%Dev_{max}$	0	0,35	0,36	0	0,13	4,46	0,17	0,09	1,98
		$\%Dev_{moy}$	0	0,03	0,14	0	0,01	0,48	0,003	0,003	0,7
	2	#Best	100	67	49	99	84	5	87	92	0
		#LB	100	67	49	99	84	5	81	86	0
		$\%Dev_{max}$	0	0,53	6,66	0,03	0,41	4,61	0,22	0,15	2,67
		$\%Dev_{moy}$	0	0,07	0,64	$3 * 10^{-6}$	0,02	0,97	0,01	0,01	1,05
	5	#Best	88	64	0	79	81	0	55	70	0
		#LB	83	58	0	64	63	0	23	29	0
		$\%Dev_{max}$	0,37	3,11	8,08	1,41	0,72	5,64	0,76	0,66	5,29
		$\%Dev_{moy}$	0,04	0,44	3,12	0,09	0,08	2,67	0,14	0,13	2,55
	7	#Best	80	60	0	55	59	0	51	53	0
		#LB	41	29	0	11	8	0	0	0	0
		$\%Dev_{max}$	1,88	5,28	9,57	2,43	2,9	9,39	1,47	1,34	6,43
		$\%Dev_{moy}$	0,35	0,62	5,11	0,51	0,51	4,67	0,45	0,44	3,93
500	1	#Best	100	90	87	100	85	12	90	100	3
		#LB	100	90	87	100	85	12	90	100	3
		$\%Dev_{max}$	0	0,03	0,58	0	0,04	1,51	0,03	0	0,75
		$\%Dev_{moy}$	0	0,003	0,01	0	0,002	0,16	0,001	0	0,16
	2	#Best	100	75	62	100	86	1	100	96	0
		#LB	100	75	62	100	86	1	99	96	0
		$\%Dev_{max}$	0	0,07	1,92	0	0,09	1,73	0,005	0,04	0,86
		$\%Dev_{moy}$	0	0,009	0,15	0	0,003	0,25	$5*10^{-7}$	$6*10^{-6}$	0,25
	5	#Best	100	80	0	97	98	0	64	79	0
		#LB	100	80	0	97	98	0	44	51	0
		$\%Dev_{max}$	0	0,33	4,03	0,1	0,01	2,63	0,19	0,14	1,55
		$\%Dev_{moy}$	0	0,03	1,22	0,001	$2*10^{-6}$	0,68	0,031	0,026	0,94
	7	#Best	84	89	0	65	70	0	53	55	0
		#LB	71	71	0	39	39	0	7	8	0
		$\%Dev_{max}$	0,25	0,25	5,08	0,58	0,8	2,98	0,48	0,41	2,79
		$\%Dev_{moy}$	0,02	0,02	1,46	0,08	0,08	1,73	0,16	0,16	1,78

TABLE 2 – Etude comparative entre les métaheuristiques pour $m = 2$.

Plus le nombre de machines augmente, plus l'écart moyen entre les méthodes et la borne inférieure augmente. Pour $m = 5$ machines, l'algorithme génétique apparaît relativement plus performant dans le cas où on a 1 et 2 types de ressources. La déviation moyenne est inférieure à $0,6\%$ pour $k = 1$ et inférieure à 4% pour $k = 2$. En effet, pour $k = 5, 7$, l'algorithme génétique hybride est en mesure de fournir des solutions meilleures que l'algorithme génétique. Par exemple, pour $n = 500$ et $k = 5$ il donne une solution réalisable dans 74 cas et une déviation moyenne inférieure à $3,3\%$ pour le premier intervalle et il trouve une solution réalisable dans 60 cas pour $k = 7$ et une déviation moyenne ne dépassant pas $3,7\%$.

4 Conclusion

Dans ce papier, nous avons considéré le problème avec un nombre arbitraire de types de ressources tel que chaque tâche nécessite soit 0 soit 1 unité de chaque type pour la phase de préparation. Ce papier a été dédié à la résolution de ce problème et ceci par des méthodes approchées basées sur des métaheuristiques. Nous avons aussi mené différents tests expérimentaux sur des instances générées aléatoirement pour évaluer la performance des métaheuristiques.

n	$\lvert R\rvert$		$s_i, p_i \in [1, 10]$			$s_i, p_i \in [10, 50]$			$s_i, p_i \in [50, 100]$		
			AG	**AGH**	**RS**	**AG**	**AGH**	**RS**	**AG**	**AGH**	**RS**
50	1	#Best	97	10	0	97	4	0	100	0	0
		#LB	75	4	0	51	0	0	62	0	0
		$\%Dev_{max}$	3,63	15,78	30,55	4,34	18,91	28,54	4,98	20,41	33,51
		$\%Dev_{moy}$	0,39	6,96	9,53	0,32	9,62	10,71	0,35	10,52	14,67
	2	#Best	90	54	0	81	36	0	91	12	0
		#LB	69	37	0	23	3	0	26	0	0
		$\%Dev_{max}$	10,31	21,42	30,83	3,37	12,85	22,06	7	13,3	29,66
		$\%Dev_{moy}$	0,76	2,43	10,6	0,69	2,8	9,23	0,81	4,34	9,68
	5	#Best	96	62	0	93	97	14	87	95	0
		#LB	45	29	0	42	43	0	23	23	0
		$\%Dev_{max}$	16,1	19,08	34,35	16,71	16,56	28,63	12,03	12,03	31,74
		$\%Dev_{moy}$	2,3	3,85	12,86	1,32	1,33	10,94	2,43	2,41	12,99
	7	#Best	98	57	0	79	91	0	88	95	0
		#LB	49	28	0	29	27	0	23	24	0
		$\%Dev_{max}$	8,72	12,79	29,86	10,71	10,96	25,45	8,8	8,7	30,03
		$\%Dev_{moy}$	1,22	2,52	9,29	1,17	1,16	9,36	1,39	1,4	9,78
100	1	#Best	98	3	0	100	1	0	100	0	0
		#LB	74	0	0	48	0	0	41	0	0
		$\%Dev_{max}$	5,02	15,0	18,72	4,41	20,45	26,79	2,62	18,17	28,66
		$\%Dev_{moy}$	0,4	8,55	9,26	0,04	11,48	11,62	0,12	12,0	14,63
	2	#Best	78	55	0	68	34	0	77	25	0
		#LB	25	10	0	8	0	0	8	0	0
		$\%Dev_{max}$	9,15	8,05	23,25	9,54	10,69	21,76	7,67	10,79	22,88
		$\%Dev_{moy}$	0,8	1,58	9,3	1,83	3,18	9,32	1,86	4,49	10,8
	5	#Best	85	86	0	70	84	0	68	85	0
		#LB	29	31	0	13	16	0	17	17	0
		$\%Dev_{max}$	11,9	11,5	36,8	14,29	14,22	24,08	11,32	11,22	25,49
		$\%Dev_{moy}$	2,75	2,33	12,9	2,35	2,32	13,2	2,02	1,9	13,19
	7	#Best	68	94	0	76	89	0	85	98	0
		#LB	13	14	0	10	11	0	6	7	0
		$\%Dev_{max}$	18,41	10,63	30,32	10,54	10,17	23,7	8,54	8,51	24,65
		$\%Dev_{moy}$	2,85	2,46	12,48	1,77	1,77	12,35	1,87	1,86	12,45
500	1	#Best	100	0	0	100	0	0	100	0	0
		#LB	18	0	0	6	0	0	0	0	0
		$\%Dev_{max}$	3,15	14,18	13,76	5,6	18,55	17,27	1,57	17,09	21,0
		$\%Dev_{moy}$	0,59	10,7	10,2	0,36	12,9	12,21	0,27	14,23	16,13
	2	#Best	43	60	0	50	50	0	70	30	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	4,52	5,19	10,39	8,39	8,33	14,59	6,82	7,47	13,56
		$\%Dev_{moy}$	3,15	3,08	7,8	4,74	4,73	9,01	4,32	5,21	11,02
	5	#Best	45	74	0	50	75	0	50	53	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	8,02	7,33	21,42	8,23	8,68	20,9	4,67	4,65	19,6
		$\%Dev_{moy}$	3,34	3,21	13,29	2,42	2,33	14,36	2,006	1,9	14,15
	7	#Best	45	60	0	34	72	0	40	74	0
		#LB	0	0	0	0	0	0	0	0	0
		$\%Dev_{max}$	8,23	8,1	21,88	8,47	7,7	22,31	7,03	6,3	23,2
		$\%Dev_{moy}$	3,74	3,61	15,63	3,13	3,0	11,53	2,73	2,52	15,9

TABLE 3 – Etude comparative entre les métaheuristiques, $m = 5$.

Références

[1] Abdelkhodaee A.H., Wirth A. Scheduling parallel machines with a single server : some solvable cases and heuristics, Computers & Operations Research, 29(3), 295–315, 2002.

[2] Abdelkhodaee A.H., Wirth A., Gan H.S. Equal processing and equal setup time cases of Scheduling parallel machines with a single server. Computers & Operations Research, 31, 1867–1889, 2004.

[3] Abdelkhodaee A.H., Wirth A., Gan H.S. Scheduling two parallel machines with a single server : the general case. Compputers & Operations Research, 33, 994–1009, 2006.

[4] Allahverdi A., Gupta J.N.D., Aldowaisan T. A review of scheduling research involving setup considerations. OMEGA The International Journal of Management Sciences, 27, 219–239, 1999.

[5] Allahverdi A., Ng C.T., Cheng T.C.E., Kovalyov M.Y. A survey of scheduling problems with setuo times or costs. European Journal of Operational Research, 187, 985–1032, 2008.

[6] Blazewicz J. Selected topics in scheduling theory. Annals of Discrete Mathematics, vol. 31, 1-61, 1987.

[7] Blazewicz J., Cellary W., Slowinski R., Weglarz J. Scheduling under resource constraints : deterministic models. Annals of Operations Research, vol. 7, 1986.

[8] Blazewicz J., Ecker K.H., Pesch E.,Schmidt G. and Weglarz J. Scheduling Computer and Manufacturing Processes. Springer-Verlag, Berlin, 2001.

[9] Blazewicz J., Lenstra J.K., Rinnooy Kan A.H.G. Scheduling subject to resource constraints : classification and complexity. Discrete Applied Mathematics, vol. 5, 11-24, 1983.

[10] Blazewicz J., Ecker K. A linear time algorithm for restricted bin packing and scheduling problems. Oper. Res. Lett., 2, 80, 1983.

[11] Blazewicz J., Kubiak W., Szwarcfiter J. Scheduling independent fixed-type tasks. In R. Slowinski and J. Weglarz (eds). Advances in Project Scheduling, Elsevier, Amsterdam, 225, 1989.

[12] Cerny, V. Thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm. J. of Optimization Theory and Applications, tome 45, n°1, P. 41-51, 1985.

[13] Holland J.H. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975.

[14] Gan HS., Wirth A., Abdelkhodaee A. A branche-and-price algorithm for the general case of scheduling parallel machines with a single server. Computer and Operations Research, 39, 2242-2247, 2012.

[15] Garey M.R., Johnson D.S. Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. 4, 397, 1974.

[16] Goldberg, D. Genetic algorithm in search, optimisation and machine learning. Addison Wesley. 36, 1989.

[17] Hasani K., Kravchenko S.A, Werner F.Block models for scheduling jobs on two parallel machines with a single server. Computer and Operations Research, 41, 94–97, 2014.

[18] Koulamas, CP. Scheduling two parallel semiautomatic machines to minimize machine interference. Computers and Operations Research, 23(10), 945–956, 1996.

[19] Kirkpatrick S., Gelatt C., Vecchi M. Optimization by simulated annealing. Science, tome 220, n°4598, P. 671–680, 1983.

[20] Kravchenko S.A., Werner F. Parallel machine scheduling problem with a single server. Mathematical and Computer Modelling, 26(12), 1–11, 1997.

[21] Labbi, W., Boudhar, M., Oulamara, A. Scheduling two identical parallel machines with preparation constraints. International Journal of Production Research, DOI : 10.1080/00207543.2014.978032. Accepted. To appear.

[22] Yang W.H., Liao C.J. Survey of scheduling research involving setup times. International Journal of Systems Science, 30, 143–155, 1999.

[23] Werner F., Kravchenko S.A. Scheduling with Multiple Servers. Automation and Remote Control, vol. 71(10), 2109–2121, 2010.