N° d'ordre :18/2018-D/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

FACULTE DE MATHEMATIQUES

THESE

Présentée pour l'obtention du grade de Docteur en Sciences

EN : MATHEMATIQUES

Spécialité : Recherche Opérationnelle (Génie Mathématiques)

Par : MEZIANI Nadjat

Thème

FLOWSHOP A DEUX MACHINES EN PRESENCE DE TACHES COUPLEES AVEC TEMPS DE LATENCE EXACTS

Soutenue publiquement le 24/06/2018, devant le jury composé de :

M. SEMRI Ahmed	Professeur	à l'U.S.T.H.B.	Président
M. BOUDHAR Mourad	Professeur	à l'U.S.T.H.B.	Directeur de thèse
M. OULAMARA Ammar	Professeur	à l'U. de Lorraine	Co-directeur de thèse
M. AITZAI Abdelhakim	Maître de Conférences/A	à l'U.S.T.H.B.	Examinateur
M. BERRICHI Ali	Maître de Conférences/A	à l'UMB Boumerdes	Examinateur
M. REBAINE Djamal	Professeur	à l'UQAC (Canada)	Examinateur

Résumé : Dans cette thèse de doctorat, nous avons étudié le problème du flowshop à deux machines dont l'objectif est la minimisation du temps total de traitement de toutes les tâches, communément appelé makespan. Dans ce modèle, les tâches sur la première machine sont constituées de deux opérations séparées par un temps de latence exact. Sur la deuxième machine, chaque tâche est constituée d'une seule opération. Nous avons montré la NP-complétude de certains cas particuliers. Pour d'autres cas, nous avons exhibé des solutions de complexité polynomiale. Pour résoudre le problème dans le cas général, nous avons opté pour l'approche heuristique. Nous avons, dans un premier temps, présenté des heuristiques basées sur les règles de priorité de type LPT et SPT et sur l'algorithme de Johnson. Dans un deuxième temps, nous avons développé deux méta-heuristiques : le recuit simulé et l'optimisation par essaims particulaires, que nous avons combiné par la suite pour proposer une solution hybride. Une étude expérimentale a été ensuite entreprise pour valider la performance de nos solutions.
Mots clès : flowshop, opérations-couplées, makespan, heuristiques, recuit simulé, optimisation par essaims de particules.

Abstract : In this doctoral thesis, we studied the flowshop scheduling problem on two-machine with coupled-task in order to minimize the makespan. Each task consists of a coupled-operation on the first machine and one operation on the second machine. Operations of each coupled-task are separated by an exact time lag. We showed the NP-hardness of some restricted versions of the studied problem, while, for some other cases, we exhibited solutions that run in polynomial time. We tackled the resolution of the general problem with heuristic approach. We first presented simple heuristics based on the LPT, SPT, and Johnson rules. We then developed two meta-heuristics : the simulated annealing and particle swarm optimization algorithms. Furthermore, we proposed a hybrid algorithm combining these two meta-heuristic. An experimental study was conducted to validate the quality and computational performances of the proposed solutions.
Keywords : flowshop, coupled-operations, makespan, heuristics, simulated annealing, PSO.

Remerciements

Le travail de recherche présenté dans cette thèse a été réalisé au sein du laboratoire RECITS de l'USTHB et du laboratoire de Lorrain de Recherche en Informatique et ses Applications (LORIA), Nancy. A ce titre, je tiens à remercier vivement Madame Marie Claude PORTMAN et Monsieur OULAMARA Ammar pour m'avoir accueilli dans ce laboratoire et à l'école des Mines de Nancy, et pour m'avoir permis d'effectuer ce travail dans les meilleures conditions.

Je tiens tout particulièrement à exprimer toute ma reconnaissance à mes directeurs de thèse, BOUDHAR Mourad, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene, et OULAMARA Ammar, Professeur à l'Université de Lorraine (France), pour avoir accepter de diriger ma thèse. Je les remercie pour le temps consacré pour moi, pour leur aide et pour leurs conseils judicieux. Qu'ils trouvent ici l'expression de ma profonde gratitude.

Je remercie Monsieur SEMRI Ahmed, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene, pour l'honneur qu'il me fait de présider ce jury.

Mes plus vifs remerciements s'adressent à Messieurs AITZAI Abdelhakim, Maître de Conférences à l'Université des Sciences et de la Technologie Houari Boumediene, et à Monsieur BERRICHI Ali, Maître de Conférences à l'Université M'hamed Bougera Boumerdès, qui ont accepté de faire partie de ce jury et d'examiner ce travail. Mes sincères remerciements vont également à Monsieur REBAINE Djamal, Professeur UQAC (Canada), pour l'intérêt qu'il a accordé à mon travail et pour l'honneur qu'il me fait de participer à ce jury.

Je tiens également à remercier les doctorants auprès de qui j'ai travaillé et qui m'ont soutenu durant mes stages à l'école des Mines. Je remercie aussi chaleureusement Madame RAMDANE CHERIF-Khettaf Wahiba, Maître de conférence à l'école des Mines, pour ses encouragements, son soutien, sa disponibilité et sa gentillesse.

Je remercie mes amis et collègues de l'U.S.T.H.B. et du laboratoire RECITS, en particulier Amina et Wafa qui m'ont aidé et soutenu. Je n'oublierai pas mes amis et collègues de travail du département de SEGC-LMD de la faculté des sciences économiques de l'université de Béjaia, je les remercie pour leur sympathie, leurs aide et soutien moral. Je remercie mes amies Zahra et Siham d'avoir été présentes pendant cette thèse, pour l'amitié et les encouragements qu'elles m'ont témoignés.

Mes derniers mots sont réservés à ma famille. Un grand merci à mes parents qui ont toujours eu confiance en moi, qui m'ont soutenu sans limite, et qui m'ont appris à surpasser les moments difficiles. Qu'ils trouvent ici le témoignage de mon affection. Merci à mon frère et mes soeurs pour leurs encouragements sans relâche et leur patience. Merci également au reste de ma famille.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

L'ordonnancement joue un rôle essentiel dans l'organisation des entreprises et de la gestion de la production. Il consiste à programmer, dans le temps, l'exécution des tâches en leur attribuant une date de début, tout en utilisant un ensemble de ressources de manière à satisfaire un ou plusieurs objectifs et en tenant compte de certaines contraintes. Les problèmes d'ordonnancement sont très variés. Ils apparaissent dans de différents domaines d'activités : l'informatique (ordonnancement de processus), l'administration (gestion des emplois du temps), l'industrie (gestion de la production) et la construction (suivi de projets). Un problème d'ordonnancement est défini par un ensemble de tâches qui requiert certaines ressources, techniques ou humaines, pour leur exécution. L'exécution de ces tâches est soumise à des contraintes, temporelles ou ressources, à respecter afin d'atteindre un objectif fixé en optimisant un ou plusieurs critères. Dans les ateliers de production, les problèmes d'ordonnancement sont classés selon le nombre de machines et leur ordre pour fabriquer un produit en plusieurs classes. Nous distinguons : problèmes à une seule machine, problèmes à machines parallèles, problèmes à machines spécialisées.

Les contraintes sont l'une des craractéristiques d'un problème d'ordonnancement et elles diffèrent d'un problème à un autre. Elles se portent sur les tâches comme sur les machines. Parmi ces contraintes, nous retrouvons les contraintes d'écarts temporels (time lags en anglais) connues également, dans la littérature, sous plusieurs dénominations, contraintes de latence, contraintes d'attente, contraintes de délais ou

contraintes de précédence généralisées. Ce type de contraintes est considéré comme une généralisation des contraintes de précédence simples. Pour ces dernières, le traitement d'une tâche ne peut commencer que si le traitement de la tâche qui la précède est terminé. Dès que la contrainte d'écarts temporels impose à la deuxième tâche de ne commencer qu'après un certain délai de la fin de la première tâche, alors le temps d'attente est minoré, ce qui défini un temps de latence minimum. Si par contre la deuxième tâche doit commencer avant un temps donné après la fin de la première tâche, l'attente est majorée, appelée temps de latence maximum. Les contraintes des temps de latence sont peu considérées dans la littérature. Elles sont introduites pour la première fois par Mitten dans les problèmes d'ordonnancement d'ateliers [92]. L'auteur a proposé une extension de l'algorithme de Johnson [61] pour résoudre le problème du flowshop de permutation à deux machines avec des temps de latence minimums. Les contraintes des temps de latence peuvent être définies de plusieurs manières : de début à début (start-start), de fin à début (stop-start) et de fin à fin (stop-stop) [21]. Nous précisons que, dans cette thèse, la contrainte de temps de latence est définie entre la date de fin de traitement d'une opération et de la date de début de la seconde opération de la même tâche.

En industrie, les contraintes des temps de latence sont très présentes. Dans certains cas, ces contraintes ne sont pas considérées, à cause de la difficulté qui réside dans la modélisation des problèmes. Deppner a exposé, dans sa thèse [41], une liste d'applications des problèmes avec la contrainte des temps de latence. Nous citons ici quelques exemples : en informatique, la tâche de sauvegarde des données nécessite l'utilisation de plusieurs opérations avec accès à des têtes d'écriture. Le nombre limité des têtes provoque l'attente de certaines opérations. Dans l'industrie chimique, un délai entre deux opérations est imposé pour éviter le refroidissement d'un produit. En agro-alimentaire, dans la production des plats surgelés, un délai maximum est nécessaire entre la cuisson des produits et leur surgélation. Dans la fabrication du papier thermique, un intervalle de temps doit être respecté pendant l'opération de stockage des rouleaux avant de passer au conditionnement. On peut également recontrer la contrainte des temps de latence dans le domaine de génie civil.

La contrainte des temps de latence est définie entre deux opérations consécutives

d'une même tâche. Dans le cas du problème à une machine, deux cas de figures sont à séparer. Dans le premier cas, chaque tâche ne contient qu'une seule opération et les contraintes de précédences sont considérées entre les tâches. Alors, des temps de latence peuvent leurs être associés. Dans le deuxième cas, chaque tâche est composée d'un certain nombre d'opérations qui sont liées par des contraintes de précédence sous forme de chaines. Ainsi, les temps de latence sont définis entre les opérations consécutives des tâches. Par conséquent, le problème de tâches couplées (coupled-task en anglais) est considéré comme un problème avec des temps de latence dans le cas où le temps de latence minimum est de même valeur que le temps de latence maximum. Ce problème a été introduit pour la première fois par Shapiro [110]. La motivation de ce problème découle du problème d'ordonnancement des tâches d'un système de radar. Chaque tâche est composée de deux opérations, la première opération concerne l'émission d'une impulssion magnétique, et la seconde, la réception de la réponse après un délai exact, c.-à-d. un temps de latence minimal égale à un temps de latence maximal. Ce problème apparait également dans des ateliers de production chimique, où une machine doit exécuter plusieurs opérations d'une même tâche, et un délai exact est imposé entre l'exécution de chaque deux opérations consécutives dû aux réactions chimiques.

Dans cette thèse, nous nous sommes intéressés à l'étude du problème du flowshop à deux machines avec des tâches couplées pour minimiser la durée totale de l'ordonnancement. Chaque tâche est composée d'un couple d'opérations sur la première machine et d'une seule opération sur la deuxième machine. Les opérations du même couple sont séparées par un délai exact.

Ce manuscrit est organisé de la manière suivante.

Le premier chapitre est consacré à la présentation des concepts de base utilisés en ordonnancement, des problèmes d'ordonnancement d'ateliers, leur classification et leur complexité, ainsi que des approches de résolution.

Dans le deuxième chapitre, nous présentons une revue de littérature sur les problèmes du flowshop avec la contrainte des temps de latence, en rappellant brièvement

les principaux travaux de recherche réalisés dans ce contexte. Nous avons distingué les cas suivants : les problèmes avec des temps de latence minimums, les problèmes avec des temps de latence maximums, les problèmes avec des temps de latence minimums et maximums, et les problèmes avec des temps de latence exacts.

Le troisième chapitre est réservé à l'étude du problème du flowshop avec des tâches couplées sous forme de deux modèles. Le premier modèle défini le problème à deux machines avec des tâches couplées sur la première machine, telle que chaque tâche est constituée de deux opérations distinctes séparées par un délai exact, et d'une seule opération sur la deuxième machine. L'objectif est de minimiser la date de fin de traitement des tâches. Le problème général étant NP-difficile, nous nous sommes donc intéressés à traiter ses sous problèmes en abordant leur complexité. Nous avons montré que certains problèmes sont NP-difficiles et autres sont résolubles en des temps polynomiaux. Le second modèle est déduit à partir du premier en inversant le rôle des tâches sur les deux machines. Les deux modèles sont symétriques. Aussi, nous terminons ce chapitre par une synthèse sur la complexité des problèmes du deuxième modèle déterminés à partir de ceux du premier modèle.

Le quatrième chapitre est dédié aux approches de résolution. Dans ce chapitre nous avons proposé plusieurs heuristiques basées sur des règles de priorités pour résoudre le problème général. Nous avons exposé également deux approches métaheuristiques, le recuit simulé et l'optimisation par essaims particulaires, pour le même problème. Par ailleurs, nous avons développé une méthode hybride en combinant les deux métaheuristiques. Enfin, nous avons présenté une étude expérimentale des méthodes proposées qui permet de vérifier leur performance.

Enfin, une conclusion dans laquelle nous présentons un bilan final de notre travail qui termine la thèse.

CHAPITRE 1

GÉNÉRALITÉS

1.1 Introduction

Dans ce chapitre, nous abordons les notions nécessaires et les concepts de base utilisés tout au long de ce travail. Nous présentons les notions préliminaires et les élèments d'un ordonnancement. Nous exposons aussi les problèmes d'ordonnancement d'ateliers, leur classification et leur complexité. Nous rappelons également les approches de résolution.

1.2 Notions de base

Ordonnancer consiste à programmer dans le temps l'exécution d'un ensemble de tâches en leur affectant un ensemble de ressources, dans le respect des contraintes fixées, afin d'optimiser un ou plusieurs critères.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des resources au cours du temps et vise à satisfaire un ou plusieurs objectifs. Plus précisément, on parle de problème d'ordonnancement lorsqu'on doit déterminer dans le temps les dates du début et de fin de traitement des tâches, alors qu'on réserve le terme de problème de séquencement

au cas où l'on cherche seulement à fixer un ordre d'exécution des tâches qui peuvent être en conflit pour l'utilisation des ressources [78].

1.2.1 Les tâches

Une tâche, appelée aussi travail (ou job en Anglais), est une entité élémentaire localisée dans le temps par une date de début t_j et de fin c_j, dont la réalisation est caractérisée par une durée p_j ($c_j = t_j + p_j$, si la tâche s'exécute sans interruption) et qui nécessite des ressources. Dans certains cas, une tâche peut s'exécuter par morceaux, la tâche est dite préemptive ou sans interruption, alors la tâche est non-préemptive. En ordonnancement d'atelier, une tâche est composée d'un ensemble d'opérations. On appelle gamme opératoire, la succession d'opérations d'une même tâche.

1.2.2 Les ressources

Une ressource k est un moyen technique ou humain requis pour la réalisation d'une tâche et disponible en quantité limitée, de capacité A_k (supposée constante). On distingue plusieurs types de ressources :

- ***Ressources renouvelables***
  Une ressource est renouvelable si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (la main d'oeuvre, les machines, l'espace, ...). La quantité de ressources utilisées à chaque instant est limitée.

- ***Ressources consommables***
  Une ressource est dite consommable si elle n'est plus disponible en même quantité après avoir été utilisée par une ou plusieurs tâches (matière première, budget, ...). La consommation globale (ou cumul) au cours du temps est limitée.

- ***Ressources doublement contraintes***
  Une ressource est doublement contrainte lorsque son utilisation instantanée

et sa consommation globale sont toutes les deux limitées (source d'énergie, financement, ...).

- ***Ressources disjonctifs***
  Une ressource est dite disjonctive ou (non partageable), si elle n'est utilisée que par une seule tâche à la fois (machine-outil, robot manipulateur).

- ***Ressources cummulatives***
  Une ressource est dite cumulative ou (partageable), si elle peut être utilisée par plusieurs tâches simultanément (équipe d'ouvriers).

1.2.3 Les contraintes

Les contraintes expriment des restrictions sur des valeurs que peuvent prendre les variables de décision. On distingue deux types de contraintes.

1. **Contraintes temporelles** : elles comprennent :

   - ***Les contraintes de potentiel :*** ces contraintes sont exprimées par l'inégalité mathématique $t_j - t_i \geq l_{i,j}$ et qui imposent une distance minimale $l_{i,j}$ entre deux instants particuliers associés aux tâches.
   - ***Les contraintes des dates limites :*** elles englobent les contraintes de localisation temporelles (dates de disponibilité) et les contraintes de délais (délais de livraison).
   - ***Les contraintes de succession ou de précédence :*** appelées aussi contraintes de cohérence technologique. Elles décrivent le positionnement relatif de certaines tâches par rapport à d'autres.

2. **Contraintes de ressources** : elles décrivent la nature et la quantité des ressources utilisées par les tâches ainsi que les caractéristiques d'utilisation de ces ressources.

1.2.4 Critères d'optimisation

Lorsqu'on aborde la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions par rapport à un ou plusieurs critères, ou à leur admissibilité vis-à-vis des contraintes. Les critères d'optimsation consistent à minimiser ou maximiser une fonction objectif. Cette dernière peut dépendre des coûts, des durées, du nombre de tâches en retard, etc.
Les critères les plus utilisés sont :

- $C_{max} = \max\{C_j / 1 \leq j \leq n\}$: la date de fin de l'ordonnancement (makespan) qui est la plus grande date de fin d'exécution des tâches.
- $\sum_{j=1}^{n} C_j$: la somme des dates de fin de traitement des tâches.
- L_{max} : la plus grande tardiveté ($L_{max} = \max\{C_j - d_j\}$ où d_j désigne la date de fin souhaitée pour la tâche j).
- $\sum_{j=1}^{n} T_j$: la somme des retards ($T_j = \max(0, C_j - d_j)$).
- $\sum_{j=1}^{n} U_j$: le nombre de tâches en retard avec $U_j = 1$ si $C_j > d_j$ et $U_j = 0$ sinon.

On note que les critères présentés ci-dessus sont tous réguliers. On dit qu'un critère est régulier si l'on ne peut pas le dégrader en avançant l'exécution d'une tâche.

1.3 Problèmes d'ordonnancement

Un ordonnancement est défini comme étant une allocation, dans le temps, des ressources disponibles aux différentes tâches à réaliser, dans le but d'optimiser un ou plusieurs objectifs. Ainsi, les ressources peuvent être des machines dans un atelier, des pistes de décollage et d'atterrissage dans un aéroport, des équipes dans un terrain de construction, des processeurs dans les ordinateurs, etc. Les tâches peuvent être des opérations dans un processus de production, le décollage et l'atterrissage dans un aéroport, les étapes d'un projet de construction, l'exécution d'un programme informatique, etc. Les différentes tâches sont caractérisées par un degré de priorité et un temps d'exécution. Les ressources, quant à elles, sont caractérisées entre autres par une capacité, des temps de réglage, etc.

Les problèmes d'ordonnancement sont généralement classés en deux principaux modèles dépendants du nombre d'opérations que requièrent les tâches : des modèles à une opération (machine unique et machines parallèles) et des modèles à plusieurs opérations dits modèles en ateliers (flowshop, job shop et open shop).

1. **Modèles à une opération :**
   - ***machine unique :*** dans un modèle à machine unique, l'ensemble des tâches à réaliser est exécuté par une seule machine.
   - ***machines parallèles :*** elles remplissent, a priori, les mêmes fonctions. Selon leurs vitesse d'exécution, on distingue :
     (a) ***machines identiques :*** la vitesse d'exécution est la même pour toutes les machines et pour toutes les tâches.
     (b) ***machines uniformes :*** chaque machine a une vitesse d'exécution propre et constante. La vitesse d'exécution est la même pour toutes les tâches d'une même machine.
     (c) ***machines indépendantes :*** la vitesse d'exécution est différente pour chaque machine et pour chaque tâche.
2. **Modèles à plusieurs opérations :** chaque tâche doit passer par plusieurs machines. En fonction du mode de passage des opérations sur les différentes machines, trois ateliers spécialisés sont différenciés, à savoir :
   - ***Flowshop :*** l'ordre d'exécution des opérations est linéaire et fixé à l'avance pour chaque tâche. Toutes les tâches ont la même gamme opératoire.
     - ***flowshop de permutation :*** la séquence des tâches visitant une machine est la même pour toutes les machines.
     - ***flowshop hybride*** : c'est une généralisation des problèmes du flowshop et de machines parallèles. L'atelier est constitué d'un certain nombre d'étages en série et chaque étage est composé de plusieurs machines en parallèles.
   - ***Job shop :*** la gamme de fabrication (l'ordre d'exécution des opérations d'une tâche sur les machines) est linéaire et fixée à l'avance, mais peut varier d'une tâche à une autre.
   - ***Open shop :*** la gamme de fabrication n'est pas fixée à l'avance.

Ces modèles sont les plus classiquement rencontrés dans la littérature, mais de

nombreux problèmes ne rentrent pas dans ce cadre (problèmes multi-machines, problèmes avec serveurs). MacCarty et Liu ont proposé une hiérarchie pour les problèmes d'ordonnancement d'ateliers de base [80].

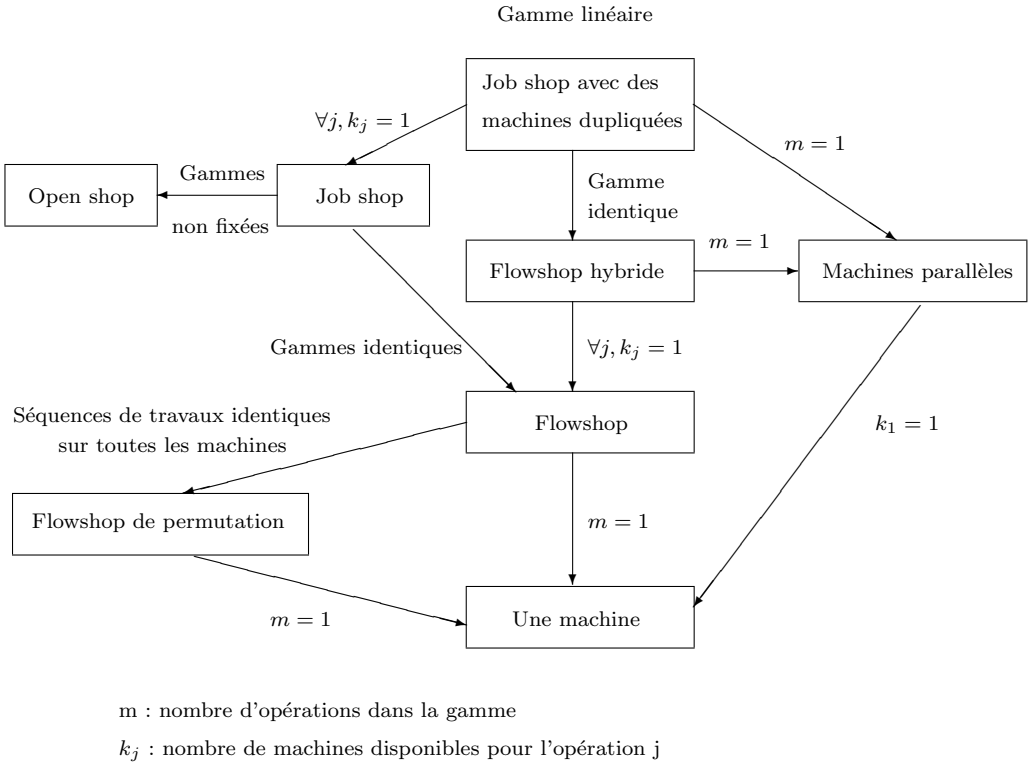

Figure 1.1 – Les problèmes d'ordonnancement d'atelier [80]

1.3.1 Classification

Il existe une très grande variété de problèmes d'ordonnancement. Pour leur identification et leur classification, nous adoptons la notation proposée dans [54]. Cette notation est constituée de trois champs $\alpha/\beta/\gamma$.

Le premier champ α est constitué de deux élèments : $\alpha = \alpha_1\alpha_2$ et décrit l'environnement des machines utilisées :

– le paramètre α_1 décrit le type des machines utilisées $\alpha_1 \in \{\emptyset,\ P,\ Q,\ R,\ F,\ J,\ O\}$ avec :

$\alpha_1 = \emptyset$: une machine ;
$\alpha_1 = P$: machines parallèles identiques ;
$\alpha_1 = Q$: machines parallèles uniformes ;
$\alpha_1 = R$: machines parallèles indépendantes ;
$\alpha_1 = F$: Flowshop ;
$\alpha_1 = J$: Job shop ;
$\alpha_1 = O$: Open shop.

– le paramètre α_2 indique le nombre de machines utilisées. Lorsque α_2 n'est pas précisé, le nombre de machines est variable.

Le deuxième champ $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7$ décrit les caractéristiques des tâches et des machines. Il indique si des élèments spécifiques sont à prendre en compte telles que les dates au plus tôt, les précédences entre tâches.

– le paramètre β_1 indique la possibilité de préemption ou non des tâches $\beta_1 \in \{\emptyset,\ pmtn\}$ avec :
$\beta_1 = \emptyset$: la préemption n'est pas permise ; une tâche commencée, doit être exécutée jusqu'à la fin ;
$\beta_1 = pmtn$: la préemption est permise, les tâches sont morcelables, elles peuvent être interrompues et terminées plus tard sur la même machine ou sur une autre machine.

– le paramètre β_2 indique la présence ou non de ressources auxiliaires $\beta_2 \in \{\emptyset,\ res\}$ avec :
$\beta_2 = \emptyset$: pas de ressources auxiliaires ;
$\beta_2 = res$: il existe des ressources auxiliaires.

– le paramètre β_3 indique la présence ou non des contraintes de précédence $\beta_3 \in \{\emptyset,\ prec,\ uan,\ tree,\ chain\}$ avec :
$\beta_3 = \emptyset$: pas de contraintes de précédence, les tâches sont indépendantes,
$\beta_3 = prec,\ uan,\ tree,\ chain$: indique respectivement des contraintes générales de précédence, des réseaux d'activité uni-connectés, des contraintes de précé-

dence formant une arborescence ou un ensemble de chaînes.

– le paramètre β_4 décrit les dates de disponibilité des tâches $\beta_4 \in \{\emptyset,\ r_j\}$ avec :
$\beta_4 = \emptyset$: toutes les dates de disponibilité sont nulles ;
$\beta_4 = r_j$: les dates de disponibilité sont différentes et sont propres à chaque tâche.

– le paramètre β_5 décrit les durées d'exécution des tâches $\beta_5 \in \{\emptyset,\ p_j = p,\ \underline{p} \leq p_j \leq \overline{p}\}$ avec :
$\beta_5 = \emptyset$: les temps opératoires sont arbitraires et sont propres à chaque tâche ;
$\beta_5 = p_j = p$: toutes les tâches ont une durée d'exécution égale à p ;
$\beta_5 = \underline{p} \leq p_j \leq \overline{p}$: toutes les tâches ont une durée d'exécution située entre $\underline{p}$ et $\overline{p}$.

– le paramètre β_6 décrit les dates d'échues au plus tard des tâches $\beta_6 \in \{\emptyset,\ d_j\}$ avec :
$\beta_6 = \emptyset$: pas de date d'échues au plus tard ;
$\beta_6 = d_j$: des dates d'échues au plus tard sont imposées.

– le paramètre β_7 indique si un temps d'attente entre les différentes opérations d'une tâche est autorisé $\beta_7 \in \{\emptyset,\ no-wait\}$ avec :
$\beta_7 = \emptyset$: un nombre illimité d'opérations peuvent être en attente devant une machine ;
$\beta_7 = no\text{-}wait$: les opérations doivent être exécutées les unes après les autres, aucune durée d'attente devant une machine n'est admise.

Le dernier champ γ indique le critère d'optimisation, il peut donc prendre de nombreuses valeurs et peut être une combinaison entre plusieurs critères : $\sum F_j$ (flot total), C_{max} (makespan) .... Par exemple, $F2/r_j/\sum C_j$ représente le problème consistant à ordonnancer n tâches dans un atelier de type flowshop, constitué de deux machines avec des dates de disponibilité, en vue de minimiser la somme des dates de fin d'exécution des tâches.

1.4 Théorie de la complexité

1.4.1 Classes P et NP

La théorie de la complexité donne une hiérarchie des complexités formelles et se base sur les problèmes de décision. Un problème de décision est un problème pour lequel la réponse est "oui" ou "non". Il est possible d'associer à chaque problème d'optimisation, un problème de décision. En particulier, un problème d'optimisation est dit NP-difficile, si son problème de décision associé est NP-complet [50].

Deux classes de complexité des problèmes de décision sont définies, sachant que l'une fait partie de l'autre :

- ***la classe P (Polynomial time)*** : la classe P regroupe les problèmes de décision qui peuvent être résolus par des algorithmes déterministes polynomiaux. Un algorithme polynomial est défini comme un algorithme dont le temps d'exécution (temps de calcul ou complexité) est borné par $O(p(x))$, où p est un polynôme et x est la longueur d'entrée d'une instance du problème.
- ***la classe NP (Non deterministic Polynomial time)*** : la classe NP regroupe les problèmes de décision qui peuvent être résolus en un temps polynomial par un algorithme non déterministe. Pour ces algorithmes, qui reposent sur des procédures particulières de choix, si à chaque instruction, le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel.

Les algorithmes polynomiaux sont évidemment des cas particuliers des algorithmes non déterministes. Aussi tout problème de décision qui peut être résolu par un algorithme polynomial, et qui appartient à la classe P, appartient également à la classe NP. D'où $P \subseteq NP$.

Parmi les problèmes de la classe NP, une large classe de problèmes, les problèmes NP-complets, sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre. C'est-à-dire, s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de

la classe NP. Afin de décrire cette classe d'équivalence, définissons tout d'abord la réduction polynomiale entre deux problèmes. Soient $P1$ et $P2$, deux problèmes de décision. On dit que $P1$ se réduit polynomialement à P2 (noté $P1 \propto P2$) s'il existe un algorithme de résolution de $P1$, qui fait appel à un algorithme de résolution de $P2$, et qui est polynomial lorsque la résolution de $P2$ est comptabilisé comme une opération élèmentaire.
On dit alors d'un problème de décision qu'il est NP-complet si tout problème de la classe NP se réduit polynomialement à lui. Les problèmes d'optimisation combinatoire dont le problème de décision associé est NP-complet sont qualifiés de NP-difficiles.

Ainsi, il est nécessaire de connaître des problèmes connus pour être NP-complets. Le premier problème qui a été prouvé comme étant NP-complet par S.A. Cook [37] est le problème de satisfiabilité (SAT) qui peut être défini comme suit : étant donnée une expression booléenne sous forme normale conjonctive (de la forme de conjonctions de disjonctions). Existe-t-il une affectation en "vrai" et "faux" de ses variables de manière à ce que l'expression booléenne prenne la valeur "vrai" ? Dans le cas où la réponse est oui l'expression sera dite satisfiable.

1.4.2 Prouver la NP-complétude d'un problème

La démonstration d'appartenance d'un problème de reconnaissance à la classe des problèmes NP-complets est très importante puisque, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème. Prouver qu'un problème de décision Π est NP-complet consiste en les trois étapes suivantes :

1. Montrer que Π est dans NP.
2. Choisir Π', un problème NP-complet connu.
3. Construire une transformation polynomiale f de $\Pi' \longrightarrow \Pi$.

1.4.3 Quelques problèmes NP-complets

Nous présentons ci-dessous quelques autres problèmes NP-complets dont la NP-complétude n'a pas pu être démontré que grâce à l'existence d'un premier problème

NP-complet. Ce sont les problèmes de base exposés par Garey et Johnson dans [50], qui détaillent ensuite la littérature concernant la complexité des problèmes.

- **Partition :** étant donné n entiers positifs $s_1,\ s_2, \ldots,\ s_n$, existe-t-il un sous ensemble $J \subseteq I = \{1,\ 2, \ldots,\ n\}$ tel que $\sum_{i \in J} s_i = \sum_{i \in I \setminus J} s_i$.

- 3**-satisfiabilité :** étant donné un ensemble de clauses de dimension 3, construites à partir d'un ensemble fini de variables, existe-t-il une affectation de ces variables qui satisfait toutes les clauses ?

- **Recouvrement :** étant donné un graphe $G = (V; E)$ et un entier k, existe-t-il $X \subseteq V$ tel que $|X| \leq k$ et toute arête de G a au moins une de ses extrémités dans X ?

- **Clique :** étant donné un graphe $G = (V; E)$ et un entier k, existe-t-il $X \subseteq V$ tel que $|X| \leq k$ et tous les sommets de X sont deux à deux adjacents ?

- **Cycle Hamiltonien :** étant donné un graphe $G = (V; E)$, existe-t-il un cycle passant une fois et une seule par chacun des sommets de V ?

1.5 Approximabilité des problèmes NP-difficiles

Les problèmes NP-difficiles de grande taille ne peuvent pas être résolus de façon optimale. Alors, il est intéressant de chercher des algorithmes qui fournissent des solutions proches de la solution optimale, et cela en s'orientant vers la construction des algorithmes polynomiaux avec garantie de performance et la construction de schémas d'approximation polynomiaux.
Dans ce qui suit, pour les définitions des notions, nous considérons le cas de problèmes de minimisation.

1.5.1 Rapport d'approximation

Le rapport d'approximation est utilisée pour évaluer la performance des méthodes de résolution. Il mesure l'erreur relative maximum d'un algorithme.

Définition. 1 *Soient X un problème d'optimisation, Π_X l'ensemble des instances de X et A un algorithme pour la résolution de X. Le rapport d'approximation de A, noté $\delta(A)$, est un nombre réel $\delta \in R$ tel que :*

$$\forall I \in \Pi_X, \quad \frac{A(I)}{OPT(I)} \leq \delta$$

où $OPT(I)$ est la solution optimale de l'instance I et $A(I)$ est la solution de l'instance I donné par l'algorithme A.

La valeur $\delta = 1 + \epsilon$, tel que $\epsilon > 0$, est dit ratio de pire cas, elle peut être intérprétée comme une mesure de qualité de l'algorithme d'approximation A ou comme une garantie de la performance de l'algorithme A. Plus δ est proche de 1 meilleur est l'algorithme.

1.5.2 Schéma d'approximation

Un schéma d'approximation est un algorithme qui, pour une précision petite donnée, apporte une solution dans la précision fixée en temps polynomial.

Définition. 2 *Soient X un problème d'optimisation et Π_X l'ensemble des instances de X. Un schéma d'approximation polynomial (polynomial time approximation scheme ou PTAS) est un algorithme AS tel que :*

$$\forall\ 0 < \epsilon < 1, \quad \forall I \in \Pi_X, \quad \frac{AS(I)}{OPT(I)} \leq 1 + \epsilon$$

où $OPT(I)$ est la solution optimale de l'instance I et $AS(I)$ est la solution de l'instance I donné par l'algorithme A en temps polynomial dépendant de la taille des entrées.

Un schéma d'approximation est également appelé un algorithme $(1+\epsilon)$-approximation.

1.5.3 Classes d'approximation

Nous présentons une classification des problèmes NP-difficiles selon le niveau possible d'approximation. Cette classification se base sur les notions d'algorithmes approchant et de schéma d'approximation polynomial et totalement polynomial.
No-APX : les problèmes pour lesquels il ne peut pas exister d'algorithme avec un

rapport d'approximation constant.
APX : les problèmes pour lesquels il existe un algorithme avec un rapport d'approximation constant.
PTAS : les problèmes pour lesquels il existe un schéma d'approximation polynomial.
FPTAS : les problèmes pour lesquels il existe un schéma d'approximation totalement polynomial.

1.6 Résolution des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont de complexité différente. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes de résolution (méthodes exactes ou heuristiques), sont largement utilisées pour appréhender les problèmes NP-difficiles. Nous exposons dans cette partie, les grandes lignes des méthodes les plus connues classées en trois catégories : les méthodes exactes, les approches heuristiques et les métaheuristiques.

1.6.1 Méthodes exactes

Dans ce paragraphe, il est question de trois types de méthodes exactes : la programmation dynamique, la méthode de séparation et évaluation et la résolution à partir d'une modélisation analytique. Le temps de calcul de ces méthodes est exponentiel ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles.

1.6.1.1 Programmation dynamique

Introduite par Bellman [14] dans les années 50, la programmation dynamique décompose un problème de dimension n en n sous problèmes de dimension 1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se faisant à partir des lois d'évolution du système et d'une décision [33].

Le principe est de procéder à une décomposition en étapes du problème, et de parcourir à rebours (de la dernière décision aux précédentes) le processus de décision séquentiel associé au problème d'ordonnancement. Chaque étape correspond à un sous problème que l'on résout optimalement en tenant compte des informations obtenues lors des étapes précédentes. Ceci nécessite une formulation du critère sous forme de relation de récurrence liant deux niveaux successifs. Comme les PSE, la programmation dynamique procède par énumération implicite de l'ensemble des solutions. C'est une méthode à vocation plus générale que la PSE, mais la taille des problèmes qu'elle permet d'aborder est en revanche plus limitée. Là encore, il est possible d'appliquer des résultats de dominance et pour des problèmes NP-difficiles de taille raisonnable, on peut ainsi construire des algorithmes de programmation dynamique pseudo-polynomiaux (désigne un algorithme capable de résoudre un problème NP-complet en un temps polynomial, moyennant un codage des données du problème en base 1).

1.6.1.2 La méthode par séparation et évaluation

Cette méthode est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Elle repose sur la construction d'un arbre à partir du problème initial qui utilise deux opérations principales : la séparation et l'évaluation. La séparation consiste à décomposer l'ensemble des solutions en plusieurs sous ensembles qui sont à leur tour décomposables d'une manière récursive. Ce processus peut se visualiser sous forme d'une arborescence d'énumération, les noeuds de l'arbre correspondent au sous ensembles et les feuilles corespondent à des solutions réalisables. La procédure d'évaluation calcule pour chaque noeud, dans le cas d'un problème de minimisation, une borne inférieure de la solution obtenue et une borne supérieure à partir de ce sous problème. Si la valeur de la borne inférieure est supérieure à la borne supérieure alors le noeud est considéré dominé et il sera éliminé. La borne supérieure est mise à jour à chaque fois qu'une solution réalisable donnant une meilleure valeur pour la fonction objectif est trouvée. La stratégie d'exploration de l'arbre correspond au choix de l'ordre d'exploration des noeuds. On distingue : la stratégie profendeur d'abord, la stratégie largeur d'abord et la stratégie meilleur d'abord. Certaines règles de dominance peuvent être définies pour permettre d'élaguer des branches de l'arbre de recherche.

1.6.1.3 Modélisation analytique et résolution

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également, parfois de le résoudre. L'idéal est d'obtenir un programme linéaire dont les variables sont réelles. Dans ce cas, il existe bon nombre de solveurs pouvant le résoudre. Il existe aussi des solveurs pour les programmes en nombre entiers (Cplex par exemple). Il en est de même lorsque le modèle n'est pas linéaire, mais quadratique par exemple. Néanmoins, il est parfois surprenant de voir qu'un problème particulier de taille raisonnable peut être appréhendé et résolu par la programmation mathématique. Il est donc justifié de commencer à étudier un problème en proposant une ou plusieurs modélisations analytiques.

Malheureusement, tous les problèmes ne peuvent être résolus par cette approche car la résolution de programmes linéaires mixtes, par exemple, demande souvent beaucoup de temps de calcul.

1.6.2 Approches heuristiques

Les approches heuristiques sont des méthodes qui fournissent des solutions de bonne qualité, c'est-à-dire proches des solutions optimales, en un laps de temps raisonnable. Ces méthodes par construction progressive sont des méthodes itératives où à chaque itération, une solution partielle est complétée. La plupart de ces méthodes sont des algorithmes gloutons car elles considèrent les élèments dans un certain ordre sans jamais remettre en question un choix, une fois qu'il a été effectué. De principe très simple, ces algorithmes permettent de trouver une solution très rapidement. Parmi ces méthodes nous en exposons deux types.

1.6.2.1 Algorithmes de liste

Le principe de ces algorithmes est d'ordonner la liste des tâches selon un ordre de priorité suivant lequel ils seront ordonnancées. Les règles de priorité les plus connues sont :

- ***SPT*** (Shortest Processing Time) : les tâches sont ordonnancées dans l'ordre croissant de leurs durées d'exécution.

- ***LPT*** (Longest Processing Time) : les tâches sont ordonnancées dans l'ordre décroissant de leurs durées d'exécution.
- ***EDD*** (Earliest Due Date) : les tâches sont ordonnancées dans l'ordre croissant de leurs dates de fin au plus tard.
- ***FIFO*** (Fist In First Out) : les tâches sont ordonnancées dans l'ordre croissant de leurs dates d'arrivées.

Malgré la simplicité de ces méthodes, elles peuvent produire de bonnes solutions.

1.6.2.2 Algorithmes gloutons

A chaque itération, une solution locale est construite pour un problème d'optimisation, dans l'espoir d'évoluer progressivement pour obtenir en résultat un optimum global. Théoriquement, ces heuristiques ne fournissent pas systématiquement à la fin de l'algorithme la solution optimale, mais elles donnent généralement des solutions très satisfaisantes.

1.6.3 Métaheuristiques

Les métaheuristiques forment une famille particulière d'heuristiques basées sur des algorithmes itératifs stochastiques visant à résoudre des problèmes d'optimisation difficiles, et qui progressent à la recherche de l'optimum global des problèmes pour lesquels on ne connaît pas de méthode classique plus efficace. Ces méthodes sont suffisamment générales pour être appliquées à plusieurs catégories de problèmes d'optimisation combinatoire. Elles peuvent être classées selon leur principe de fonctionnement dans la recherche de la solution en deux catégories : les métaheuristiques à base de voisinage (à solution unique) et métaheuristiques à base d'une population de solutions.

1.6.3.1. Métaheuristiques à base de voisinage

Ces méthodes sont initialisées par une solution réalisable, calculée soit aléatoirement soit à l'aide d'une heuristique constructive et recherchent à chaque itération

une amélioration de la solution courante par des modifications locales. Cet examen se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. L'utilisation de ces heuristiques itératives suppose que l'on puisse définir pour toute solution un voisinage de solution contenant les solutions voisines.

1.6.3.1.1 Méthode de descente

Elle consiste à rechercher dans le voisinage de la solution courante, une solution de coût plus faible. Elle procède ainsi jusqu'à arriver à un optimum local. Cette méthode a l'avantage d'être rapide, mais s'arrête dès qu'un optimum local est atteint, même si celui-ci n'est pas de bonne qualité.
Parmi ces méthodes, nous décrivons ici les méthodes d'échanges de type r-opt. Ces méthodes d'optimisation ont été initialement proposées par Lin dans [76] pour résoudre le Problème du Voyageur de Commerce (PVC), mais elles s'appliquent également à tout problème combinatoire dont la solution consiste en une permutation de r composantes parmi n.

Le terme r-optimal indique qu'une solution ne peut plus être améliorée en échangeant r élèments. La méthode consiste donc à sélectionner r composantes et à regarder si en interchangeant ces composantes, on obtient une meilleure solution. Nous remarquons donc qu'une solution n-optimale est une solution optimale (dans le cas d'un problème de taille n). Ainsi, plus r augmente, plus on se rapproche de la solution optimale, mais plus les calculs sont difficiles. En effet, il y a C_n^r façons de choisir r composantes et $r!$ façons de les échanger. En pratique, on se limite à $r = 2$ ou 3.

1.6.3.1.2 Recherche avec tabous

La méthode de recherche avec tabous est basée sur des mécanismes inspirés de la mémoire humaine [43]. Son principe de base est simple : la méthode tabou fonctionne avec une seule configuration courante à la fois. Au départ une seule solution quelconque qui est actualisée au cours des itérations successives. A chaque itération, le mécanisme de passage d'une configuration, soit s, à la suivante, soit t, comporte deux étapes :

– On construit l'ensemble des voisins de s c'est à dire l'ensemble des configurations accessibles en un seul mouvement élèmentaire à partir de s. Soit $v(s)$

l'ensemble de ces voisins,

– On évalue la fonction objectif f du problème en chacune des configurations appartenant à $v(s)$. La configuration t, qui succède à s dans la suite des solutions construites par la méthode tabou est la configuration de $v(s)$ en laquelle f prend la valeur minimale. Notons que cette configuration t est adoptée même si elle est moins bonne que s, c'est à dire si $f(t) > f(s)$.

Pour éviter de retourner à une configuration déjà retenue lors d'une itération précédente, on tient à jour et on exploite à chaque itération une liste de mouvements interdits, la "liste des tabous" ou "liste taboue".

1.6.3.1.3 Recuit simulé

Ie recuit simulé est inspiré du domaine de la thermodynamique et s'appuie sur les méthodes de simulation de Metropolis [68]. La technique du recuit en métallurgie consiste à porter le matériau à une haute température puis à abaisser lentement celle-ci, ce qui permet d'obtenir un état solide bien ordonné d'énergie minimale. La méthode du recuit simulé transpose le procédé du recuit à la résolution d'un problème d'optimisation : la fonction objectif du problème, analogue à l'énergie d'un matériau, est alors minimisée, moyennant l'introduction d'une température fictive qui est dans ce cas un simple paramètre de contrôle de l'algorithme.

1.6.3.2. Métaheuristiques à base de population

Connues aussi sous l'appellation d'algorithmes évolutionnistes, sont des méthodes de recherche qui utilisent une population de solutions qui interragissent entre eux afin de produire de nouveaux individus plus performants. Ces métaheuristiques sont inspirées de la biologie et utilisant des phénomènes d'auto organisation. Parmi ces méthodes on trouve :

1.6.3.2.1. Optimisation par essaims de particules

L'optimisation par essaims de particules (OEP) est une méthode d'optimisation stochastique, pour les fonctions non linéaires, dévellopé par Kennedy et Heberhat [65]. Elle s'inspire du comportement collectifs des déplacements d'animaux. Cette méthode est une procédure de recherche basée sur une population d'individus, ap-

pelés particules, qui changent leurs positions avec le temps. Dans le système d'OEP, les particules se déplacent à l'intérieur d'un espace de recherche et chaque particule ajuste sa trajectoire vers sa meilleure position dans le passé et vers la meilleure position des particules de son voisinage. La variante globale de ces algorithmes considère la totalité de l'essaim comme voisinage. Les particules profitent ainsi des découvertes et expériences antérieures de toutes les autres particules.

1.6.3.2.2. Algorithmes génétiques

Cette classe de méthodes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. L'application de ces méthodes aux problèmes d'optimisation a été formalisée par Goldberg en 1989 dans [53].

Les algorithmes génétiques fonctionnent sur une analogie avec la reproduction des êtres vivants. On part d'une population (ensemble de solutions) initiale sur laquelle des opérations de reproduction, de croisement ou de mutation vont être réalisées dans l'objectif d'exploiter au mieux les caractéristiques et propriétés de cette population. Ces opérations doivent mener à une amélioration (en terme de qualité des solutions) de l'ensemble de la population puisque les bonnes solutions sont encouragées à échanger par croisement leurs caractéristiques et à engendrer des solutions encore meilleures. Toutefois, des solutions de très mauvaises qualités peuvent aussi apparaitre et permettent d'éviter de tomber trop rapidement dans un optimum local. Les difficultés pour appliquer les algorithmes génétiques résident dans le besoin de coder les solutions, et dans les nombreux paramètres à fixer (taille de la population, coefficients de reproduction, probabilitée de mutation, ...). De plus, ces algorithmes demandent un effort de calcul très important.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

2.1 Introduction

Ce chapite présente une revue de la littérature qui couvre les principaux résultats traitant les problèmes d'ordonnancement de type flowshop. Comme nous nous intéressons au problème du flowshop avec des tâches couplées en présence de la contrainte des temps de latence, alors, nous exposons les principaux travaux réalisés avec la contrainte des temps de latence minimums, des temps de latence maximums, des temps de latence minimums et maximums et des temps de latence exacts.

2.2 Problèmes du flowshop

Le problème du flowshop, $F//C_{max}$, est considéré comme l'un des problèmes les plus étudié dans la littérature. Pour le problème à deux machines, Johnson dans [61], a établi un algorithme polynomial qui a servi de base à de nombreuses méthodes de résolution qu'elles soient des méthodes exactes, d'heuristiques ou de bornes inférieures pour des problèmes NP-difficiles. Le problème devient NP-difficile au sens fort [51], si le nombre de machines est supérieur à deux.

Johnson a montré que toute séquence respectant la règle $\min(p_{i,1}, p_{j,2}) \leq \min(p_{j,1},$

$p_{i,2}) \Longleftrightarrow$ (la tâche i est traitée avant la tâche j) est optimale, et a proposé un algorithme de complexité $O(nlogn)$. L'algorithme est le suivant :

Algorithme. Johnson

début

- *Construire les ensembles des tâches* $X = \{T_j / p_{1j} \leq p_{2j}\}$ *et* $T \setminus X = \{T_j / p_{1j} > p_{2j}\}$,
- *Ordonnancer sur les deux machines les tâches de* X *suivant l'ordre croissant de leurs temps de traitement sur la première machine,*
- *Ordonnancer sur les deux machines les tâches de* $T \setminus X$ *suivant l'ordre décroissant de leurs temps de traitement sur la deuxième machine.*

fin

Plusieurs se sont intéressés à étudier le problème du flowshop de permutation, parmi lesquels nous citons les travaux de Campbell et al. [26], Dannenbring [38], Taillard [115] et Widmem [120]. En se servant du résultat de Johnson [61], Bellmann et al. [13] ont proposé un algorithme qui permet d'énumérer toutes les séquences satisfaisant la règle de Johnson. Aussi dans [15], Billaut et Lopez ont suggéré un algorithme qui énumère toutes les séquences optimales.

Ramon et Manelo ont proposé, dans [101], une méthode exacte pour résoudre les problèmes du flowshop, $F/prmu/C_{max}$ et $F/block/C_{max}$, et des approches heuristiques. Ces approches sont utilisées pour trouver une solution initiale et s'appliquent en deux étapes. La première étape consiste à trouver une permutation tandis que la seconde étape est réservée à l'application de la méthode de la recherche locale. Lomniki [77] et Ignall et al. [59] ont utilisé la première méthode exacte, Branch and Bound, pour résoudre le problème $F/prmu/C_{max}$. Plus tard, d'autres auteurs ont proposé des méthodes de ce type pour résoudre le même problème [8],[27],[35],[99].

Un autre type de problèmes du flowshop pour lequel les auteurs ont porté un intérêt particulier est le problème du flowshop à deux machines, $F2/no-wait/C_{max}$, avec la contrainte sans attente. La contrainte *no-wait* exprime qu'aucune attente entre deux opérations d'une même tâche n'est permise. Ce problème est résolu en

temps polynomial par l'algorithme de Gilmore et Gomory [52]. Dans [97], les auteurs ont traité le problème du flowshop de permutation avec la contrainte *no-wait*. Ils ont modélisé le problème $F/no-wait/C_{max}$ par un réseau où la longueur du chemin critique représente le makespan. Ils ont donné une réduction du problème au problème du voyageur de commerce, comme ils ont présenté encore quelques sous cas polynomiaux.

La contrainte no-idle décrit qu'il n'existe aucun temps mort sur une machine entre le début de son activation et la fin de l'exécution des opérations qui lui sont affectées. Adiri et Pohoryles [1] sont les premiers à avoir étudiés le problème du flowshop à m machines et ils ont montré que le problème $F2//C_{max}$ est équivalent au problème $F2/no-idle/C_{max}$. Baptiste et Hguny [11] se sont intéressés au problème du flowshop avec la contrainte no-idle. Ils ont prouvé que le problème $F3/no-idle/C_{max}$ est NP-difficile au sens fort et ont développé une méthode de type Branch and Bound pour résoudre le problème $F/no-idle/C_{max}$. Saadani et al. [108] ont proposé une heuristique en $O(nlogn)$ pour résoudre ce problème avec trois machines, comme ils ont évalué sa performance en comparant leurs résultats avec la borne inférieure, aussi avec la solution trouvée en utilisant le logiciel LINGO pour des instances de petites tailles. Les mêmes auteurs, dans [107], ont proposé une procédure, pour résoudre le problème $F/no-idle/C_{max}$, basée sur l'idée que le problème peut être modélisé comme un problème du voyageur de commerce.

Dans [97], Pawel et Kamburowski ont modélisé le problème $F/no-idle/C_{max}$ par un réseau où la longueur critique du chemin représente le makespan. Dans [63], Kamburowski s'est intéressé à l'étude du problème $F3/no-idle/C_{max}$. Il a identifié le problème par une simple représentation du réseau et a décrit certaines anomalies provenant de la contrainte no-idle, comme il a déterminé certaines relations de dominance sous lesquelles le problème est résolu efficacement.

Espinouse et al. [45] ont étudié le même problème en présence de la contrainte d'indisponibilité sur l'une des machines. Ils ont montré que les problèmes $F2, h_{11}/no-wait/C_{max}$ et $F2, h_{21}/no-wait/C_{max}$ sont NP-difficiles. Avec au moins deux périodes d'indisponibilité séparées et même si ces périodes se situent sur une même machine,

ils ont montré que le problème $F2, h_{jk}/no-wait/C_{max}$ est NP-difficile au sens fort. En se basant sur l'algorithme de Gilmore et Gomory [52], ils ont également proposé des heuristiques polynomiales avec un ratio de performance égale à 2.

Wang et Cheng [119] se sont intéressés au même problème avec une période d'indisponibilité sur une seule machine pour lequel ils ont proposé deux heuristiques avec un ratio de performance égale à $\frac{5}{3}$. Cependant, Cheng et Liu [30] ont proposé un schéma d'approximation en temps polynomial (PTAS) avec une seule période d'indisponibilité sur l'une des machines ou sur les deux machines ensemble. Pour les deux problèmes, les même auteurs ont proposé dans [29] un ratio de performance qui est égale à $\frac{3}{2}$.

Riad Aggoune [5] a abordé le problème du flowshop, $F, h_{ik}//C_{max}$, à plusieurs machines avec des périodes d'indisponibilité imposées dues à la maintenance. Il a considéré deux variantes non préemptives du problème. Dans la première variante, les dates de début des travaux de maintenance sont fixées et dans la seconde, les travaux de maintenance doivent être exécutés sur des fenêtres temporelles données. Il a également proposé des heuristiques basées sur l'algorithme génétique et la recherche tabou. Aggoune et Portman [4] proposent, pour le même problème, un algorithme d'approximation basé sur une extension des approches géométriques.

Lee dans [72] a traité le problème du flowshop à deux machines avec la contrainte d'indisponibilité pour minimiser le makespan. L'auteur a prouvé que les problèmes $F2/r-a(M1)/C_{max}$ et $F2/r-a(M2)/C_{max}$ sont NP-difficiles en présence d'une période d'indisponibilité sur l'une des machines. Il a proposé un algorithme de programmation dynamique pour résoudre le problème optimalement et des heuristiques en $O(nlogn)$. La première heuristique résout le problème avec indisponibilité sur la première machine avec un ratio de performance égale à $\frac{3}{2}$ au pire des cas, et la deuxième heuristique résout le problème si l'indisponibilité est définie sur la deuxième machine avec un ratio de performance égale à $\frac{4}{3}$.

Le même auteur a abordé les mêmes problèmes avec des tâches résumables et non résumables dans [71]. Il a donné la complexité des problèmes définis dans [72]

et a présenté aussi des heuristiques pour leur résolution. Cheng et Wang, dans [31], se sont intéressés au problème $F2/r - a(M1)/C_{max}$ étudié par Lee [72], et ils ont développé une autre heuristique avec un ratio de performance égale à $\frac{4}{3}$. D'autre part, Breit [20] a traité le même problème $F2/r - a(M1)/C_{max}$ en présence de l'indisponibilité sur la deuxième machine, et il a proposé une heuristique avec un ratio de performance égale à $\frac{5}{4}$.

Blazewicz et al. ont étudié dans [17] le problème du flowshop à deux machines avec plusieurs périodes d'indisponibilité. Ils ont développé des heuristiques et une métaheuristique (recuit simulé) pour résoudre le problème. Le même problème avec des tâches résumables a été abordé par Kubiak et al. dans [70]. Ils ont prouvé que les problèmes $F2, h_{1k}/r - a/C_{max}$ et $F2, h_{2k}/r - a/C_{max}$ sont NP-difficiles au sens fort. Pour la résolution du problème $F2, h_{jk}/r - a/C_{max}$, ils ont proposé une borne inférieure, des propriétés de dominance et une méthode exacte de type Branch and Bound.

2.3 Problèmes avec des temps de latence minimaux

Un temps de latence minimum est un délai qui est défini entre deux opérations consécutives de la même tâche. Cette contrainte est exprimée généralement en fonction de la date de fin de traitement d'une opération C_{1j} et de la date de début de traitement de l'opération suivante t_{2j}. Elle est donnée par : $C_{1j} + l_j^{min} \leq t_{2j}$. Les temps de latence minimaux peuvent être associés à des contraintes d'écart minimal particulières tels que les temps de séchage, de refroidissement, les temps de réglage ou les temps de montage. Suivant la valeur du temps de latence, nous distinguons trois cas :

1. $l_j^{min} > 0$: après l'exécution de la première opération d'une tâche, la tâche doit attendre un temps l_j^{min} pour traiter l'opération suivante. Ce type de contrainte est utilisée pour modéliser un temps de transport entre deux opérations consécutives. Ainsi, le délai l_j^{min} correspond au temps du transport de la tâche (voir figure 2.1).

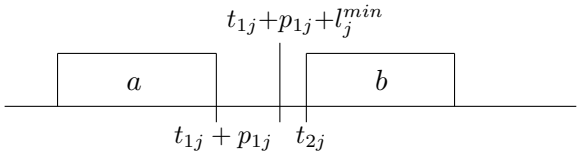

Figure 2.1 – Contrainte de temps de latence minimum : $l_j^{min} > 0$

2. $l_j^{min} < 0$: la durée du temps de latence minimum est négative. Il est possible également de rencontrer la contrainte des time lags dans les ateliers de production par lots. Après le traitement d'une partie d'une tâche du lot sur une machine, il est possible d'anticiper son exécution sur la machine suivante avant la fin de son exécution sur la machine précédente. Ainsi, les deux opérations consécutives qui modélisent le traitement du lot se chevauchent. Dans ce cas, les temps de latence minimaux de type start-start représentent le temps de traitement d'un lot sur la première machine, tandis que ceux de type stop-stop représentent le temps de traitement d'un lot sur la seconde machine. Kim [67] utilise la contrainte de temps de latence dans ce contexte.

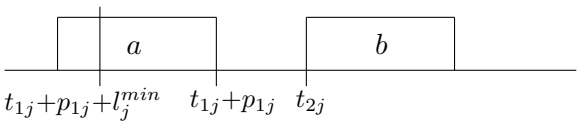

Figure 2.2 – Contrainte de temps de latence minimum : $l_j^{min} < 0$

3. $l_j^{min} = 0$: dans le cas où la valeur du temps de latence est nulle, cela signifie qu'il n'existe pas de temps d'attente entre deux opérations consécutives, ce qui exprime une contrainte de précédence classique.

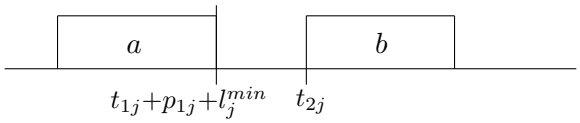

Figure 2.3 – Contrainte de temps de latence minimum : $l_j^{min} = 0$

Le premier problème considéré dans la littérature avec un temps de latence minimum est le problème du flowshop de permutation à deux machines pour minimiser le

makespan, $F2/perm, l_j^{min}/C_{max}$, étudié par Mitten [92]. Il a prouvé que ce problème peut être résolu en temps polynomial, en $O(nlogn)$, en appliquant l'algorithme de Johnson [61] pour résoudre le problème du flowshop à deux machines, $F2//C_{max}$, avec les nouvelles durées de traitement $p'_{1j} = p_{1j} + l_j^{min}$ et $p'_{2j} = p_{2j} + l_j^{min}$.

Les chercheurs ne se sont pas limités au problème du flowshop de permutation. Kern et Nawijn, dans [66], ont traité le problème d'ordonnancement à une machine, $1/l_j^{min}/C_{max}$, où chaque tâche est composée de deux opérations séparées par un temps de latence minimum dans le but de minimiser le makespan. Ils ont prouvé que ce problème est NP-difficile.

Dans [126], Yu et al. ont montré que le problème $1/l_j^{min}/C_{max}$ est équivalent au problème à deux machines avec un délai minimum $F2/l_j^{min}/C_{max}$. Cependant, ils ont prouvé que le problème à deux machines avec un délai minimum et des durées opératoires unitaires, $F2/p_{ij} = 1, l_j^{min}/C_{max}$, est NP-difficile au sens fort. Ils ont montré également que le problème à une machine avec un délai minimum, $1/p_j = 1, l_j^{min}/C_{max}$, est aussi NP-difficile [126]. D'autre part, si les temps de latence ne peuvent prendrent que deux valeurs différentes, $l_j^{min} \in \{l_1, l_2\}$, alors le problème $F2/l_j^{min} \in \{l_1, l_2\}, p_{1j} = p_{2j}/C_{max}$ reste NP-difficile au sens fort [127].

Dans [73], Lenstra a montré que le problème du flowshop à deux machines avec un délai minimum, $F2/l_j^{min}/C_{max}$, est NP-difficile au sens fort et dans un autre travail [39], Dell'Amicco considère le même problème. Il a montré qu'il est NP-difficile au sens fort, y compris dans sa version préemptive $F2/pmtn, l_j^{min}/C_{max}$. Il a présenté plusieurs bornes inférieures avec leurs ratios de performance dans le pire des cas. Outre les bornes simples obtenues à partir de la charge des machines et de la durée totale des travaux, d'autres bornes sont calculées en relachant la contrainte de capacité sur l'une des machines ou en réduisant les temps de latence. A partir des séquences obtenues par relaxation, des solutions réalisables sont construites fournissant des bornes supérieures avec des ratios de performance. Pour la résolution du problème, l'auteur a développé une méthode basée sur une recherche tabou. L'auteur avec Vaessens dans [40], ont prouvé que le problème $F2/l_j^{min}, p_{1j} = p_{2j}/C_{max}$ est NP-difficile au sens fort aussi.

Gupta, dans son papier [55], a abordé le problème à une machine, $1/O_j = 2, l_j^{min}/C_{max}$, telle que chaque tâche est composée de deux opérations séparées par un délai minimum pour minimiser le makespan et il a prouvé que le problème est NP-difficile au sens fort. Il a présenté des bornes inférieures et des heuristiques basées sur des règles de priorités ou sur l'algorithme de Johnson [61]. Ces approches génèrent dans un premier temps des ordonnancements de permutation, puis appliquent des procédures d'amélioration. D'après les expérimentations numériques, l'efficacité de ces heuristiques croit avec le nombre de tâches.

Finta et Liu, dans [46], considèrent le problème à une machine avec des dates de disponibilité, queues et temps de latence minimaux positifs entre les tâches. Ils ont montré comment utiliser ce type de contraintes pour traiter l'indisponibilité de la machine, puis ils ont prouvé que le problème est NP-difficile pour des durées opératoires unitaires. Si par contre les temps de latence minimaux sont tous égaux à 1, le problème est résoluble en temps polynomial, comme ils se sont intéressés au cas non préemptif.

Janczewski et kubal, dans [60], se sont intéressés aux problèmes à une machine avec des opérations de durées unitaires, et en présence de temps de latence uniquement minimaux et symétriques dans le but de minimiser le makespan. Les contraintes sont également présentées sous forme d'un graphe pondéré. Les auteurs ont prouvé que le problème est équivalent au problème de T-Coloration et à la minimisation du T-Span. Grâce à l'équivalence entre les deux problèmes, les auteurs déduisent des résultats de complexité pour le problème d'ordonnancement, en fonction de la structure du graphe. Ils ont montré qu'il est polynomial si le graphe est bipartie, pseudo-polynomial si le graphe a un nombre cyclomatique borné et NP-difficile au sens fort si le graphe est complet.

Lin et Haley [75] ont étudié le problème de la minimisation du makespan sur une machine où chaque tâche est composée de deux opérations séparées par un temps de latence minimal, $1/O_j = 2, l_j^{min}/C_{max}$, pour minimiser la plus grande date de fin de traitement des tâches. Ils ont proposé un programme mathématique en variables mixtes pour modéliser le problème et ont présenté un certain nombre de propriétés :

- Propriété 1 : les ordonnancements dans lesquels les secondes opérations des tâches sont traitées après toutes les premières opérations, celles-ci étant exécutées sans temps mort sont dominants.
- Propriété 2 : pour une séquence donnée des premières opérations, il suffit de traiter les secondes opérations selon leurs dates de disponibilité. Le problème revient donc à déterminer l'ordre optimal d'exécution des premières opérations.
- Propriété 3 : le problème étudié est équivalent au problème $F2/l_j^{min}/C_{max}$.

Les auteurs ont présenté également des cas particuliers résolubles en des temps polynomiaux et ont développé une méthode exacte pour résoudre le problème général, dans laquelle ils ont utilisé des heuristiques qu'ils ont proposé afin d'obtenir une borne supérieure.

Balas et al. ont étudié, dans [9], le problème à une machine avec des dates de disponibilités, queues et temps latence minimaux positifs entre les tâches pour minimiser la date de fin de traitement des tâches. Ils ont prouvé que le problème est NP-difficile au sens fort même si toutes les dates de disponibilités d'une part, et toutes les queues, d'autre part sont égales, que ce soit dans le cas non préemptif ou dans le cas préemptif.

Munier et Sourd, dans [95], ont considéré le problème de minimisation de la date de fin de traitement des tâches sur une machine, $1/chaine(l_{ij})/C_{max}$, en présence de temps de latence minimaux et des contraintes de précédence sous forme de chaîne. Ils ont prouvé que le sous problème $1/chaine(l), p_j = p/C_{max}$ est résolu en temps polynomial, lorsque les durées des opérations sont fixes et les temps de latence sont également constants. Cependant, ils se sont intéressés au cas où la valeur de n'importe quel temps de latence est inférieure à la valeur minimale des durées de traitement, d'où le problème $1/chaine(l_{ij} \leq p_h)/C_{max}$, pour lequel ils ont proposé un algorithme polynomial en $O(nlogn)$ pour minimiser le makespan. Pour minimiser la somme des dates de fin de traitement des tâches, avec des durées de traitement constantes, $1/chaine(l_{ij} \leq p_h), p_j = p/\sum C_j$, le problème peut être résolu en temps polynomial en $O(n^2)$ grâce à la programmation dynamique.

Brucker et al. [22] ont fourni des résultats complémentaires. Ils ont proposé un algorithme pseudo polynomial pour résoudre le problème de minimisation de la somme des dates de fin de traitement des tâches sur une seule machine, $1/chaine(l)$, $p_j = p/\sum C_j$, lorsque les durées d'une part et les temps de latence minimaux d'autre part sont constants, en utilisant la structure particulière des ordonnancements optimaux. En outre, ils ont montré que cet algorithme peut être ramener à un algorithme polynomial, grâce à une description compacte des solutions pour résoudre le même problème pour minimiser le makespan.

Dans [79], Lourenço H.R. a considéré le problème à une machine avec préemption des tâches et en présence des temps de latence sous forme de chaîne. Chaque tâche possède une date de disponibilité et une date de début de traitement au plutôt (delivery time). Le but est de minimiser la date de la plus grande tardiveté (maximum lateness L_{max}). L'auteur a proposé un algorithme polynomial pour résoudre le problème qui consiste à appliquer la version préemptive de l'heuristique. Cependant, il est utilisé pour obtenir des bornes inférieures pour les problèmes à une machine et du job shop avec des temps de latence.

Rebaine, dans son papier [104], a abordé le problème du flowshop de permutation sur deux machines avec des temps de latence minimaux pour minimiser la date de fin de traitement des tâches. Il a prouvé que le rapport de performance entre le flowshop de permutation et la solution optimale est borné par 2. Lorsque toutes les durées opératoires des tâches sont unitaires, le ratio est réduit à $2 - 3/(n + 2)$, et à m quand il s'agit d'un flowshop à m machines.

Rayward et Rebaine ont abordé dans [103] le problème du flowshop avec délai afin de minimiser la date de fin de traitement des tâches. Ce délai est considéré comme étant le temps du transport d'une tâche entre deux machines consécutives. Les auteurs ont montré que ce problème est NP-difficile même si les durées opératoires sont unitaires.

Dans [94], Munier et Rebaine se sont intéressés au problème du flowshop de permutation avec des durées de traitement unitaires et en présence de délais arbitraires.

Leur objectif est de minimiser la date de fin de traitement des tâches. Cependant, ils ont développé des algorithmes polynomiaux pour les problèmes à trois et à quatre machines, en $O(nlogn)$ et $O(n^2)$, respectivement.

Rayward et Rebaine ont étudié [102] le problème du flowshop à deux machines avec des temps de latence minimaux et des durées d'exécution unitaires, pour minimiser le makespan. Ils ont proposé deux heuristiques pour résoudre le problème avec des ratios de performance. Ils ont aussi réalisé des expérimentations numériques pour étudier la performance de ces heuristiques.

Moukrim et al. [93] ont étudié le problème du flowshop à deux machines avec des délais minimums. Les durées de traitement considérées sont unitaires et l'objectif est de minimiser le makespan. Les auteurs ont proposé une méthode exacte de type Branch and Bound pour la résolution du problème. D'autre part, ils ont proposé des bornes inférieures et supérieures, ainsi que des règles de dominance. D'après les expérimentations numériques réalisées, l'algorithme fournit une solution optimale pour le problème.

Khurana et Bagga [69] ont étudié le problème du flowshop de permutation à deux machines avec des temps de latence minimaux et en présence des temps de réglage, dans le but de minimiser le makespan. Ils ont traité le cas spécial des temps latence où ces délais stat-start et stop-stop sont égaux, comme ils ont présenté des conditions sur les temps de réglage en affirmant que, si elles sont vérifiées, elles permettent de construire un ordonnancement optimal.

Le travail présenté par Wlodzimierz dans [122] généralise le modèle du flowshop proposé par Khurana et Bagga dans [69]. L'auteur a étudié le problème du flowshop en présence des temps de latence et des temps de setup lorsqu'aucune restriction n'est imposée pour minimiser le makespan. Ils ont développé une approche qui fournit une solution optimale pour résoudre le problème à deux machines et une solution approximative dans le cas de plusieurs machines, comme ils ont proposé une borne inférieure pour le makespan.

Dans [23], Brucker et al. ont étudié le problème du flowshop avec des temps de transport dans l'objectif de minimiser la somme des dates de fin de traitement des tâches. Ce dernier est équivalent au problème du flowshop avec des temps de latence minimaux entre les opérations appartenant à la même tâche. Les auteurs ont prouvé que le problème du flowshop avec les temps du transport et des durées de traitement constantes, $F/p_{ij} = p, t_j/\sum C_j$, est polynomial en $O(nlogn)$, et se résout en rangeant les tâches dans l'ordre SPT des temps du transport sur toutes les machines. Pour minimiser la somme pondérée des dates de fin de traitement des tâches, et si les poids et les temps du transport sont agreeable (en anglais) (on dit que les temps de transport t_j et les poids w_j sont agreeable si $w_1 \geq w_2 \geq \ldots \geq w_n$ et $t_1 \leq t_2 \leq \ldots \leq t_n$), alors il existe un ordonnancement optimal pour le problème $F/p_{ij} = p, t_j/\sum w_j C_j$, en rangeant les tâches dans l'ordre SPT des temps du transport de toutes les machines, comme ils ont prouvé que les problèmes $F2/p_{ij} = 1, t_j/\sum w_j C_j$ et $F2/p_{ij} = 1, t_j, r_j/\sum C_j$ sont NP-difficiles au sens fort en utilisant une réduction de 3-partition.

Brucker et Knust [25] ont montré que les problèmes préemptifs à m machines parallèles avec des contraintes de précédence se réduisent aux problèmes à une machine avec des durées opératoires unitaires, et des temps de latence minimaux constants égales à $m-1$. Pour les problèmes à machines parallèles, non preemptifs avec des contraintes de précédence sous forme de intree et outtree, et avec des durées de traitement unitaires pour minimiser le L_{max}, ils ont prouvé qu'ils se réduisent aux problèmes à une seule machine avec des durées opératoires unitaires et des temps de latence minimaux constants. Ils ont montré également que les problèmes d'ordonnancement à une seule machine avec des contraintes de précédence se réduisent aux problèmes analogues avec des temps de latence minimaux de valeurs fixes. En outre, ils ont proposé des algorithmes polynomiaux pour résoudre les problèmes à une machine, de durées opératoires unitaires et des temps de latence minimaux constants, $1/p_j = 1, outtree(l), r_j/\sum C_j \quad \text{et} \quad 1/p_j = 1, prec(l)/\sum C_j$, pour minimiser la somme des dates de fin de traitement des tâches. Enfin, ils ont présenté une preuve de complexité pour le problème $1/chains(l)\ /\sum C_j$, où ils ont démontré que ce problème est NP-difficile au sens fort.

Riezebos et al. [106] ont abordé le problème du flowshop à m machines avec plusieurs opérations par tâche et temps de latence minimaux pour minimiser le makespan. Chaque tâche j possède $N_{j,k}$ opérations qui doivent être exécutées sur la machine k, cependant le nombre d'opérations est différent d'une tâche à une autre et d'une machine à une autre. Les temps de latence minimaux sont définis entre chaque couple d'opérations, que ces opérations soient affectées à la même machine ou pas. Ce problème est issu des systèmes de production flexibles, pour lesquels les produits subissent plusieurs traitement sur une même station de travail, séparés par des temps de transport et d'autres activités effectuées sur des ressources non critiques. Ils ont formulé le problème sous forme d'un programme mathématique et ont proposé plusieurs bornes inférieures. Des heuristiques par construction sont également présentées, certaines utilisent les bornes inférieures pour la sélection des opérations à ajouter.

Riezebos et Gaalman ont proposé, dans [105], la suite du travail proposé dans [106]. Les auteurs ont étudié l'effet de la taille des temps de latence sur la performance des heuristiques qui sont construites à partir d'une autre heuristique basée sur une borne inférieure pour le makespan. L'efficacité de ces heuristiques a été évaluée de façon empirique pour différentes valeurs de la taille des temps de latence, en résolvant un grand nombre de problèmes générés aléatoirement. Ils ont prouvé que si le rapport entre la valeur moyenne des temps de latence et la durée moyenne de traitement est de 20% ou plus, alors les heuristiques générant des ordonnancements sans délai sont meilleures que celles qui génèrent des ordonnancements actifs et l'inverse est vrai, si ce rapport est inférieur à 20%. Ils ont également testé et comparé ces heuristiques avec d'autres heuristiques, et ils ont noté que les heuristiques faisant appel aux bornes inférieures sont beaucoup plus efficaces que les heuristiques par construction de base, qui se reposent sur des règles simples de priorité.

2.4 Problèmes avec des temps de latence maximaux

Dans le cas où le temps d'attente entre deux opérations consécutives d'une tâche ne doit pas dépasser une valeur donnée, la contrainte de temps de latence est maxi-

male. Elle est également exprimée en fonction des dates de fin C_{1j} et de début t_{2j} des deux opérations qui se succèdent. Elle est décrite par : $C_{1j} + l_j^{max} \geq t_{2j}$. Les contraintes des temps de latence maximums permettent de modéliser un délai maximum entre deux opérations.

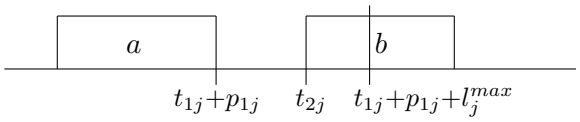

Figure 2.4 – Contrainte du temps latence maximum

Les exemples sont multiples :

- dans les processus thermiques ou chimiques : temps limité entre deux opérations pour éviter le refroidissement d'un produit.
- dans l'industrie agro-alimentaire : avec l'existence de produits périssables et de normes sanitaires.
- dans le domaine d'agriculture : avec les contraintes portant sur l'enchaînement des différents travaux lors des récoltes.

Les problèmes avec les contraintes des temps de latence maximaux sont considérés comme des cas intermidiaires entre deux situations extrêmes : les problèmes classiques, sans temps de latence maximaux ($l_j^{max} = +\infty$), tel que les cas du flowshop à deux machines pour minimiser le makespan, résolu par l'algorithme de Johnson, et le flowshop sans attente ($l_j^{max} = 0$) pour lesquels les opérations successives des tâches doivent se succéder immédiatement, sans temps d'attente. Les travaux de recherche se portent surtout sur les temps de latence minimaux et ceux qui traitent les temps de latence maximaux sont peu nombreux.

Yang et Chern, dans [125], ont montré que le problème de permutation à deux machines avec un temps de latence maximum, $F2/perm, l_j^{max}/C_{max}$, pour minimiser le makespan est NP-difficile au sens ordinaire. Pour sa résolution, ils ont développé une procédure par séparation et évaluation avec des expérimentations numériques. Plus tard, Fondrevelle et al. [48] ont prouvé que le même problème avec un délai

maximum constant est NP-difficile au sens fort.

2.5 Problèmes avec des temps de latence minimaux et maximaux

Dans [98], Potts et Whitehead ont étudié le problème à une machine dans lequel chaque tâche consiste en deux opérations séparées par un délai borné par un temps de latence minimal et un temps de latence maximal dont l'objectif est de minimiser le makespan. Le problème étant NP-difficile. L'évaluation d'une solution pour toute permutation d'opérations proposée nécessite un algorithme, de complexité $O(n^2)$, qui calcule le plus long chemin dans un graphe. Parmi les algorithmes proposés, nous retrouvons des algorithmes de descente et des méthodes taboues réactives, c'est-à-dire pour lesquelles la taille de la liste des solutions taboues varie afin de privilégier l'intensification dans certains cas et la diversification dans d'autres. Une méthode approchée par construction est développée en deux versions, déterministe et aléatoire. Les auteurs ont affirmé que les expériences menées montrent qu'une application répétée de cette heuristique sous sa forme aléatoire est plus performante que les méthodes de recherche locale, particulièrement de type tabou.

Dans [111], Sheen et Liao ont traité le problème d'ordonnancement à une machine avec un délai minimum et maximum pour minimiser le makespan qu'ils ont modélisé par un graphe disjonctif. Ils ont décrit la méthode proposée par Carlier et Pinson [28] de type Branch and Bound, dans laquelle ils ont introduit le concept de input et de output pour une clique donnée ce qui améliore l'efficacité de la manière d'élaguer les noeuds de branchement non réalisables en considérant la contrainte de délai.

Chu et Proth [34] ont étudié le problème à une machine pour minimiser le makespan, tout en respectant les contraintes de délais minimaux et maximaux formant des chaînes. La difficulté de ce problème réside dans l'obtention d'une solution faisable, et ils ont précisé que la relaxation lagrangienne ne peut pas être appliquée pour ce problème, du fait des temps de latence maximaux. Une preuve de NP-complétude est présentée. Pour sa résolution, trois heuristiques sont proposées et une méthode

de type Branch and Bound avec des expérimentations numériques.

Dans [121], Wikum et al. se sont intéressés à l'étude du problème à une machine en présence des contraintes de précédence généralisées. Ils ont considéré à la fois des temps de latence minimaux et maximaux, mais ils se sont limités au cas où ces contraintes sont sous forme de chaines. Leur objectif est de minimiser le makespan. Pour des relations de précédence simple, ils ont présenté trois cas particuliers polynomiaux et pour des structures plus complexes, ils ont fourni des résultats de NP-complétude. Pour l'un des cas NP-difficile où seulement des temps de latence minimaux sont considérés, et toutes les chaines sont composées d'une seule tâche sauf une qui est composée de deux tâches, une heuristique est proposée avec un ratio de performance de 3/2.

Hurink et Keuchel, dans [57], ont étudié le problème d'ordonnancement à une machine avec des délais positifs (minimums) et négatifs (maximums) dans le but de minimiser le makespan, qu'ils ont résolu en utilisant la méthode de recherche locale. La difficulté de l'application de cette méthode réside dans la détermination de la solution de départ. Cela est réglé en intégrant des solutions irréalisables dans le processus de recherche de solutions.

Brucker et al. [24] ont montré que les problèmes complexes d'ordonnancement, tels que les problèmes d'ateliers généralisés, les problèmes avec des tâches multi-processeurs, les problèmes avec machines à usages multiples, et les problèmes avec temps de changement, peuvent être réduit aux problèmes à une seule machine avec un délai minimum et maximum entre les tâches. Une méthode de type Branch and Bound a été développée pour résoudre ce problème à une machine pour minimiser le makespan. Par ailleurs, les résultats obtenus montrent que les méthodes de résolution des problèmes d'ateliers, qui utilisent la transformation et l'algorithme de Branch and Bound utilisés pour résoudre celui à une seule machine, ne sont pas aussi efficaces que les méthodes appliquées directes aux problèmes d'ateliers.

Fondrevelle et al., dans [47], ont traité le problème du flowshop avec un délai minimum et maximum dans le but de minimiser la somme des dates de fin de trai-

tement pondérées. Ils ont montré que ce problème généralise le même problème pour minimiser le makespan et déduisent la complexité des problèmes du flowshop à deux et à trois machines. Ils ont proposé une méthode de type Branch and Bound pour la résolution du problème du flowshop de permutation à m machines. Les auteurs ont étudié dans [48] le problème du flowshop de permutation à deux machines, $F2/perm, l_j^{min}, l_j^{max}/C_{max}$, avec un délai minimum et (ou) maximum dans l'objectif de minimiser la date de fin de traitement de toutes les tâches. Ils ont prouvé que le problème de permutation à deux machines avec un délai maximum constant est NP-difficile au sens fort. Ils ont développé une méthode exacte de type Branch and Bound pour résoudre le problème de permutation à m machines avec un délai minimum et maximum. Ils ont également testé plusieurs bornes inférieures et des heuristiques qui fournissent des bornes supérieures. D'autre part, Bouquard et Lenté dans [18], ont proposé une autre méthode de type Branch and Bound pour résoudre le même problème à deux machines. Pour plus de détails, Fondrevelle a donné dans sa thèse [49], une étude approfondie du problème du flowshop avec des temps de latence minimaux et maximaux, et un état de l'art sur les contraintes des temps de latence.

Dans [42], Deppner considère les problèmes d'ordonnancement d'ateliers plus généraux, avec des gammes de fabrication quelconques, des contraintes de calendrier sur les machines, et des temps de latence minimaux et maximaux. Son objectif est de concevoir des méthodes de résolution efficaces capables de fournir des solutions respectant l'ensemble des contraintes, pour des instances issues de situations réelles. Cependant, il a développé un algorithme de construction basé sur le placement d'opérations par grappes à partir d'un algorithme de liste de priorité et qui considère notamment les temps de latence minimaux et maximaux. Dans le cadre du flowshop de permutation et sous certaines hypothèses, l'algorithme proposé permet de construire une solution valide en temps polynomial. Pour la résolution du problème, un algorithme génétique qui fait appel à plusieurs opérateurs de croisement est conçu avec une méthode de recherche locale pour améliorer les résultats [41].

2.6 Problèmes avec des temps de latence exacts

En ordonnancement, le problème de tâches couplées avec un délai exact est un cas particulier des problèmes avec des temps de latence, dans le cas où le délai qui sépare deux opérations de la même tâche correspond à un temps de latence minimal et un temps de latence maximal de même valeur. Il a été introduit pour la première fois par Shapiro [110] afin de modéliser les problèmes de transmission par ondes d'un radar. Il a constaté que le problème à une machine avec un délai exact pour minimiser le makespan, $1/p_{1j}, l_j, p_{2j}/C_{max}$, est équivalent au problème du job shop sans attente à deux machines, dans lequel une tâche est composée de trois opérations et la seconde machine n'est pas goulet. Comme il a proposé des heuristiques avec des expérimentations numériques pour la résolution du problème.

Orman et Potts [96] ont également étudié le même problème de tâches couplées à une machine avec un délai exact, c-à-d. en présence des temps de latence minimaux et maximaux égaux ($l_j^{min} = l_j^{max} = l_j$) pour minimiser le makespan. Ils ont prouvé qu'il existe une équivalence entre chaque problème de minimisation du makespan et son problème inverse. Cependant, Ils ont abordé plusieurs problèmes, et ils ont démontré que certains sont résolubles en des temps polynomiaux et autres sont NP-difficiles, comme ils ont considéré le problème $1/\textit{tâche-couplée}, p_{1j} = a, l_j = l, p_{2j} = b/C_{max}$ comme un problème ouvert.

D'autres auteurs se sont intéressés à l'étude du problème laissé ouvert par Orman et Potts [96]. Ahr et al. [6] ont proposé un algorithme exact utilisant la programmation dynamique de complexité de $O(nr^{2L})$ où $r \leq \sqrt[a-1]{a}$ tel que l'algorithme est linéaire en nombre de tâches pour un L fixé. Les auteurs définissent une technique de représentation utilisant des patterns composés de 1 et de 0 et représentant, respectivement, les temps d'utilisation du processeur et les temps d'inactivité. Ils ont décrit alors, selon les valeurs de a et b, des graphes orientés pondérés représentant tous les patterns possibles. Puis, l'algorithme parcourt les différents patterns selon les arcs pour obtenir le poids minimum représentant le makespan. Cette programmation dynamique permet de résoudre le problème pour des petites instances, lorsque L est fixé. Cet algorithme a été adapté par Brauner et al. dans [19] pour résoudre

le problème de tâches couplées motivé par les problèmes de gestion de temps de production cyclique avec des robots.

Dans le but de minimiser le makespan, Békési et al. [12] ont étudié le problème de tâches couplées sur une machine en présence de délai exact entre deux opérations de chaque tâche avec des durées opératoires unitaires. Ils ont présenté également de nouvelles bornes inférieures.

D'autres chercheurs se sont dirigés vers des algorithmes d'approximation pour ces problèmes. Ainsi, Ageev et Baburin [2] ont proposé des algorithmes de 7/4 et 3/2-approximation pour résoudre les problèmes de tâches couplées sur une seule et deux machines avec des durées de traitement unitaires notés par, $1/\textit{tâche-couplée}, p_{1j} = p_{2j} = 1, l_j/C_{max}$ et $F2/\textit{tâche-couplée}, p_{1j} = p_{2j} = 1, l_j/C_{max}$, respectivement. L'objectif est de minimiser le makespan. En outre, ils ont prouvé que ces algorithmes sont exécutés en $O(nlogn)$ et $O(n^2logn)$ pour le premier et le second problème, respectivement.

Pour les problèmes $1/p_{1j}, l_j, p_{2j}/C_{max}$ et $F2/p_{1j}, l_j, p_{2j}/C_{max}$, Ageev et Kononov [3] ont fourni plusieurs résultats d'approximation et des bornes de non approximabilité selon les valeurs de p_{1j} et p_{2j}. Alors, ils ont présenté des algorithmes de 3.5-approximation pour résoudre le problème à une machine et 3-approximation pour résoudre ses sous problèmes lorsque $p_{1j} \leq p_{2j}$ ou $p_{1j} \geq p_{2j}$. Ils ont montré aussi que cet algorithme fournit une 2.5-approximation lorsque $p_{1j} = p_{2j}$. Pour le problème $F2/p_{1j}, l_j, p_{2j}/C_{max}$, les auteurs ont donné un algorithme de 3-approximation pour le cas général et ils ont prouvé que cet algorithme fournit 2-approximation dans les cas où $p_{1j} \leq p_{2j}$ ou $p_{1j} \geq p_{2j}$. Ces algorithmes sont implémentés en $O(nlogn)$. Ils ont prouvé également l'existence d'un algorithme de (2-ϵ)-approximation pour une seule machine et d'un autre de (1.5-ϵ)-approximation pour le problème à deux machines. Les résultats de non-approximabilité, obtenus avec les algorithmes proposés par les auteurs, montrent que dans le cas où les opérations de chaque tâche ont la même durée de traitement, les rapports d'approximation sont possibles pour les deux problèmes.

Baptiste, dans [10], a étudié le problème de tâches couplées sur une machine, $1/\textit{tâche-couplée}, p_{1j} = a, l_j = l, p_{2j} = b/C_{max}$, pour minimiser le makespan lorsque les durées de traitement p_{1j}, p_{2j} et le délai l_j sont fixes ($p_{1j} = a, p_{2j} = b, l_j = l$). Il a prouvé que le problème peut être résolu en $O(logn)$ en se basant sur les résultats obtenus par Ahr et al. [6].

Peu de travaux ont été réalisés en rajoutant des contraintes aux tâches couplées. Blazewicz et al. [16] ont prouvé que le problème $1/\textit{tâche-couplée}, p_{1j} = p_{2j} = 1, l_j = l/C_{max}$ est NP-difficile au sens fort, en ajoutant une contrainte de précédence sous forme d'un graphe général entre les tâches couplées pour obtenir le problème noté par $1/(1, l, 1) - coupled, strict\ prec, exact\ gap/C_{max} \leq D$. Comme ils ont proposé un algorithme qui peut résoudre le problème avec un $l = 2$, en $O(n)$, sous des contraintes de précédence sous forme de intree ou outtree.

Yu et al. [126] ont montré que le problème du flowshop à deux machines avec des temps de latence exacts et des durées opératoires unitaires noté par $F2/p_{1j} = p_{2j} = 1, l_j/C_{max}$ est NP-difficile.

Hwang et al. dans [58] ont considéré le problème d'ordonnancement de tâches couplées avec un délai exact sur une seule machine, soumise à une séquence fixée de tâches pour minimiser le makespan. Comme ce problème reste toujours un problème ouvert, les auteurs ont proposé un algorithme en $O(n^2)$ pour trouver la valeur minimale du makespan pour une permutation de $2n$ tâches soumises à cette contrainte. Ils ont également suggéré quelques sous problèmes résolubles en des temps polynomiaux avec des séquences de tâches fixes, et ils ont présenté le graphe de complexité des différents problèmes étudiés.

Condotta et al. ont élaboré une métaheuristique de type recherche tabou pour résoudre le problème de tâches couplées sur une seule machine avec des durées et des délais exacts quelconques dont l'objectif est de minimiser le makespan [36]. Afin d'explorer le maximum de solutions, ils ont étudié le cas spécial du problème avec une permutation fixe de la première opération des tâches et ils ont prouvé qu'il est NP-difficile pour des durées opératoires unitaires. L'algorithme de recherche tabou

utilisé repose sur deux approches pour améliorer la recherche de la solution : (1) maintenir l'ensemble des solutions classées selon les évaluations des améliorations possibles qui peuvent être réalisées si un voisin est généré ; (2) créer une liste taboue qui mémorise les caractéristiques des solutions éliminées sur les chemins critiques.

Simonin et al. [113] ont étudié le problème de tâches couplées soumises à des contraintes de compatibilité sur une seule machine sur laquelle deux types de tâches doivent être traitées, à savoir les tâches d'acquisition et les tâches de traitement. Ce problème est motivé par le problème d'acquisition de données par une torpille en immersion. Une sonde emet une onde qui se propage sous l'eau pour recueillir des données, appelées tâches d'acquisition. Ainsi, deux sous tâches sont obtenues, une qui envoie l'echo et l'autre qui le reçoit, et séparées par un temps d'inactivité qui représente la propagation de l'écho sous l'eau. Pendant ce temps d'inactivité, une autre tâche peut être réalisée, ces tâches sont dites tâches de traitement. Les auteurs ont proposé un algorithme polynomial pour résoudre le problème $1/prec, (p_{1j} = l_j = p, p_{2j}) \cup (\tau_j, pmtn), G_c/C_{max}$. Dans [112], ils ont prouvé que le problème $1/prec, (p_{1j} = p_{2j} = p, l_j = l) \cup (\tau_j, pmtn), G_c/C_{max}$ est NP-difficile, comme ils ont développé un algorithme d'approximation avec une garantie de performance basé sur un couplage maximum. Dans sa thèse [114], Simonin a donné plus de détails sur ces problèmes.

Joseph et al. [62] ont traité le problème du flowshop à deux machines avec un délai exact entre deux opérations de chaque tâche en considérant les deux objectifs : minimiser le makespan et la somme des dates de fin de traitement des tâches. Ils ont opté pour l'utilisation de la notation suivante : $F2^g$ pour représenter le modèle du flowshop à deux machines dans le cas où la tâche peut avoir une ou deux durées opératoires non nulles, et $F2$ pour représenter le même problème dans le cas où la tâche possède exactement deux durées non nulles. Pour le premier critère, ils ont prouvé que le problème $F2/l_j \in \{l_1, l_2\}/C_{max}$ est NP-difficile même si les délais prennent seulement deux valeurs différentes. Sans donner de preuve, ils ont noté que le problème $F2/l_j = l/C_{max}$ peut être résolu en $O(nlogn)$ et dans un autre théorème, ils ont ennoncé qu'un ordonnancement optimal peut être construit en $O(nlogn)$ pour le problème $F2/l_j \geq p_{1,j+1} + l_{j+1} + p_{2,j+1}/C_{max}$. D'autre part, ils

se sont intéressés à l'étude des algorithmes d'approximation du problème général et quelques sous problèmes. Alors, ils ont développé un algorithme de 3-approximation pour le problème $F2^g/l_j/C_{max}$ en $O(nlogn)$. De même pour les sous problèmes $F2^g/l_j, p_{1j} \leq p_{2j}/C_{max}$ et $F2^g/l_j, p_{1j} \geq p_{2j}/C_{max}$, ils ont proposé deux autres algorithmes de 2-approximations en $O(nlogn)$. Pour ce qui est du second objectif, ils ont proposé un algorithme en temps polynomial pour résoudre le problème $F2/l_j, p_{1j} = p_1, p_{2j} = p_2, p_1 \geq p_2/\sum C_j$ et un algorithme de 2-approximation en $O(nlogn)$ pour le problème $F2/l_j, p_{1j} = p_1, p_{2j} = p_2, p_1 < p_2/\sum C_j$.

Dans [56], Huo et al. ont étudié le problème du flowshop de permutation à deux machines $F2/perm, l_j/\sum C_j$, avec des tâches couplées et des délais exacts dans le but de minimiser la somme des dates de fin de traitement des tâches. Ce problème est une généralisation du problème à deux machines sans attente, noté par $F2/no-wait/\sum C_j$, connu pour être NP-difficile lorsque $l_j = 0$. Ils ont prouvé qu'une solution optimale est obtenue avec la règle SPT pour les problèmes $F2/perm, l_j, p_{1j} = p_1, p_{2j} = p_2/\sum C_j$, $F2/l_j = l, p_{1i} < p_{1j} \rightarrow p_{2i} < p_{2j}/\sum C_j$ et $F2/no-wait, p_{1i} < p_{1j} \rightarrow p_{2i} < p_{2j}/\sum C_j$. Deux métaheuristiques, la recherche taboue et le recuit simulé, avec des heuristiques ont été élaborées. Des tests numériques ont été réalisées pour résoudre le problème général.

Amrouche et Boudhar ont étudié, dans [7], le problème du flowshop à deux machines avec réentrance et en présence des temps de latence exacts, noté par $F2/chaine, reen-trance, l_j = l/C_{max}$. L'objectif est de minimiser la date de fin de traitement des tâches. Dans ce problème, chaque tâche doit passer sur les deux machines M_1 et M_2 dans cet ordre $M_1 \rightarrow M_2 \rightarrow M_1$, et un délai exact est imposé entre la date de fin de traitement de la première opération sur la première machine et de la date de début de traitement de la deuxième opération sur la même machine. Ils ont montré que ce problème est NP-difficile au sens fort. Ils ont également présenté des sous problèmes résolus en des temps polynomiaux.

Prblèmes du flowshop		
Problème	Complexité	Référence
$F2//C_{max}$	Polynomial	Johnson [61]
$F3//C_{max}$	NP-difficile	Garey et al. [51]
$F2/no-wait/C_{max}$	Polynomial	Gilmore et Gomory [52]
$F3/no-idle/C_{max}$	NP-difficile	Baptiste et Hguny [11]
$F2,h_{11}/no-wait/C_{max}$	NP-difficile	Espinouse et al. [45]
$F2,h_{21}/no-wait/C_{max}$	NP-difficile	Espinouse et al. [45]
$F2/r-a(M_1)/C_{max}$	NP-difficile	Lee [71]
$F2/r-a(M_2)/C_{max}$	NP-difficile	Lee [71]
$F2,h_{1k}/r-a/C_{max}$	NP-difficile	Kubiak et al. [70]
$F2,h_{2k}/r-a/C_{max}$	NP-difficile	Kubiak et al. [70]

Table 2.1 – Complexité des problèmes du flowshop

Temps de latence minimaux		
Problème	Complexité	Référence
$F2/perm,\ l_j^{min}/C_{max}$	Polynomial	Mitten [92]
$F2/p_{1j}=p_{2j}=1,l_j^{min}/C_{max}$	NP-difficile	Yu et al. [126]
$F2/l_j^{min}\in\{l_1,l_2\},p_{1j}=p_{2j}/C_{max}$	NP-difficile	Yu et al. [127]
$F2/l_j^{min}\in\{l_1,l_2\},p_{1j}=p_{2j}=1/C_{max}$	Polynomial	Yu et al. [127]
$F2/l_j^{min}/C_{max}$	NP-difficile	Lenstra [73]
$F2/pmtn,l_j^{min}/C_{max}$	NP-difficile	Dell'Amicco [39]
$F2/l_j^{min},p_{1j}=p_{2j}/C_{max}$	NP-difficile	Dell'Amico et al. [40]
$F2/p_{ij}=1,t_j/\sum w_jC_j$	NP-difficile	Brucker et al. [22]
$F2/p_{ij}=1,t_j,r_j/\sum C_j$	NP-difficile	Brucker et al. [22]
$1/O_j=2,l_j^{min}/C_{max}$	NP-difficile	Gupta [55]
$1/O_j=2,l_j^{min}/C_{max}$	NP-difficile	Lin et Haley [75]
$1/p_{1j}=p_{2j}=1,l_j^{min}/C_{max}$	NP-difficile	Yu et al. [126]
$1/O_2=2,l_j^{min}/C_{max}$	NP-difficile	Kern et Nawijn [66]

Table 2.2 – Complexité des P.F.S. avec des temps de latence minimaux

Problème	Complexité	Référence
$F2/perm, l_j^{max}/C_{max}$	NP-difficile	Yang et Chern [125]
$F2/perm, l_j^{max} = l/C_{max}$	NP-difficile	Fondrevelle et al. [48]

Table 2.3 – Complexité des P.F.S. avec des temps de latence maximaux

Problème	Complexité	Référence
$F2/perm, l_j^{min}, l_j^{max}/\sum W_j C_j$	NP-difficile	Fondrevelle et al. [47]

Table 2.4 – Complexité des P.F.S. avec des temps de latence minimaux et maximaux

Problème	Complexité	Référence
$1/$tâche-couplée$, p_{1j} = l_j = p_{2j}/C_{max}$	NP-difficile	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j}, l_j = l, p_{2j} = b/C_{max}$	NP-difficile	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j} = a, l_j = l, p_{2j}/C_{max}$	NP-difficile	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j} = p_{2j} = p, l_j/C_{max}$	NP-difficile	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j} = a, p_{1j} = b, l_j/C_{max}$	NP-difficile	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j} = l_j = p, p_{2j}/C_{max}$	Polynomial	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j}, l_j = p_{2j} = p/C_{max}$	Polynomial	Orman et Potts [96]
$1/$tâche-couplée$, p_{1j} = p_{2j} = p, l_j = l/C_{max}$	Polynomial	Orman et Potts [96]
$F2/l_j \in \{l_1, l_2\}/C_{max}$	NP-difficile	Joseph et al. [62]
$1/(1, l, 1) - coupled, strict$	NP-difficile	Blazewicz et al. [16]
$prec, exact\ gap/C_{max} \leq D$		
$F2/p_{1j} = p_{2j} = 1, l_j/C_{max}$	NP-difficile	Yu et al. [126]
$1/prec, (p_{1j} = p_{2j} = p, l_j = l)$	NP-difficile	Simonin et al. [112]
$\cup(\tau_j, pmtn), G_c/C_{max}$		
$F2/perm, l_j/\sum C_j$	NP-difficile	Huo et al. [56]
$F2/chaine, reentrance, l_j = l/C_{max}$	NP-difficile	Amrouche et Boudhar [7]

Table 2.5 – Complexité des P.F.S. avec des temps de latence exacts

CHAPITRE 3

PROBLÈME DU FLOWSHOP AVEC DES TÂCHES COUPLÉES

3.1 Introduction

Dans ce chapitre, nous étudions le problème du flowshop à deux machines avec des opérations couplées dans l'objectif de minimiser le makespan. Cependant, nous distinguons deux modèles. Le premier modèle est le problème du flowshop avec des opérations couplées sur la première machine et une seule opération sur la deuxième machine, tandis que le second modèle est le problème du flowshop avec une seule opération sur la première machine et un couple d'opérations sur la deuxième machine. Pour les deux modèles, chaque deux opérations du même couple sont séparées par un délai exact. Comme l'objectif est de minimiser le makespan, alors les deux modèles sont équivalents. A partir de là, nous nous sommes intéressés à étudier la complexité des problèmes du premier modèle. Ainsi, nous montrons que certains sous problèmes sont NP-difficiles et autres sont polynomiaux, et nous proposons également des algorithmes pour résoudre les problèmes polynomiaux. Pour terminer cette étude, nous donnons la complexité des problèmes pour le second modèle.

3.2 Description du problème

Le problème d'ordonnancement de tâches couplées a été introduit pour la première fois par Shapiro [110]. Une tâche couplée est composée de deux opérations distinctes qui s'exécutent dans l'ordre séparées par un délai, un temps de latence ou un time lag exact. Chaque tâche couplée est notée par le triplet (a_j, L_j, b_j), représentant le temps d'exécution de la première opération (a_j), l'intervalle du temps qui s'écoule entre la date de fin de traitement de la première opération et de la date de début de traitement de la deuxième opération (L_j), et le temps d'exécution de la deuxième opération (b_j). Pendant le temps de latence, la machine est inactive et une opération ou les deux opérations (voir la figure 3.1) d'une autre tâche peuvent être traitées. Le but est de minimiser la date de fin de traitement des tâches (C_{max}) ou autres critères réguliers.

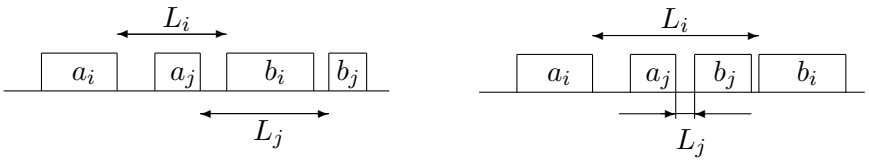

Figure 3.1 – Exemples d'entrelacement de tâches

Dans le cas d'un problème à une seule machine, toutes les opérations sont traitées sur la machine. Orman et Potts [96] ont noté ce problème par $1/a_j, L_j, b_j/C_{max}$. Le problème d'ordonnancement avec des temps de latence exacts est également abordé dans le cas du flowshop. Le problème à deux machines avec un délai exact consiste à ordonnancer un ensemble de tâches, où chaque tâche est composée de deux opérations. Pour chaque tâche, la première opération est exécutée sur la première machine, tandis que la deuxième opération est exécutée sur la deuxième machine, après un délai exact de la date de fin de traitement de la première opération. Ce problème est noté par $F2/a_j, L_j, b_j/C_{max}$. La motivation de ce problème découle d'un problème d'ordonnancement des tâches d'un radar qui consiste dans l'émission des impulsions et la réception des réponses après un temps d'attente. Ce problème apparaît aussi dans des ateliers de production chimique, où une machine doit exécuter plusieurs opérations d'une même tâche et un délai exact est imposé entre l'exécution de chaque deux opérations consécutives dues aux réactions chimiques.

Dans ce travail, nous considérons le problème du flowshop avec des opérations couplées. Ce problème consiste à traiter un ensemble de n tâches, telle que chaque tâche est composée de deux couples d'opérations, $O_{1,j}$ et $O_{2,j}$, qui doivent être exécutées sur la première et la seconde machine, respectivement. Les deux opérations d'un même couple d'opérations sont séparées par un délai exact. Les couples d'opérations $O_{1,j}$ et $O_{2,j}$ sont définis par leurs durées de traitement et leurs délais qui sont donnés par les triplets (a_j, L_j, b_j) et (c_j, H_j, d_j), respectivement. Le traitement du second couple d'opérations sur la deuxième machine ne commence qu'après l'achèvement du traitement du premier couple d'opérations sur la première machine.

Cependant, nous considérons deux modèles du problème d'ordonnancement des couples d'opérations sur deux machines. Dans le premier modèle, modèle 1, chaque tâche est composée d'un couple d'opérations qui s'exécutent sur la première machine et d'une seule opération sur la deuxième machine. Dans le second modèle, modèle 2, les tâches sont composées d'une seule opération sur la première machine et d'un couple d'opérations sur la deuxième machine. L'objectif est de minimiser la date de fin de traitrement des tâches (makespan), ce qui résulte que les deux modèles sont symétriques.

Dans [96], Orman et Potts ont traité le problème de tâches couplées sur une seule machine en présence des temps de latence exacts entre deux opérations de chaque tâche pour minimiser le makespan. Ils ont prouvé que le problème est équivalent au même problème en inversant les rôles des opérations sur la machine. Ils ont déterminé de nombreux résultats dépendants des valeurs de a_j, L_j et b_j. Ainsi, ils ont donné une classification de ces problèmes suivant leur complexité, à savoir : problèmes NP-difficiles, problèmes résolubles en des temps polynomiaux et problèmes ouverts. Notre travail s'inspire des résultats trouvés par ces auteurs. D'autre part, il est clair que tout problème NP-difficile sur une seule machine est également NP-difficile dans le cas du problème du flowshop. Notre étude se focalise sur les problèmes prouvés déjà résolubles en des temps polynomiaux dans le cas d'une seule machine. La figure (3.2) représente les problèmes de cette classe proposée par Orman et Potts [96].

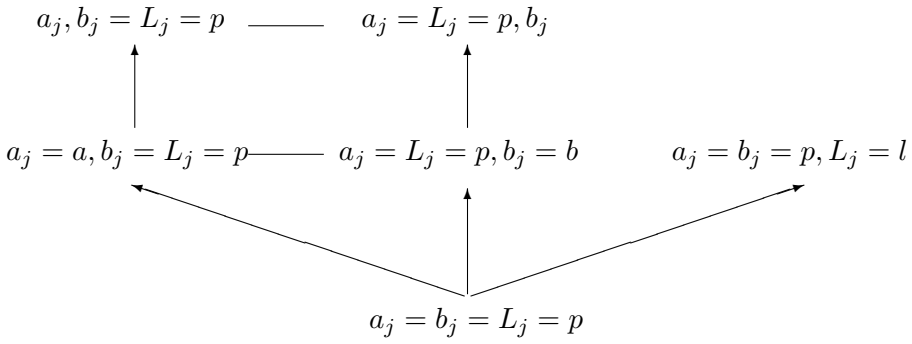

Figure 3.2 – Problèmes résolubles en des temps polynomiaux-Orman et Potts [96]

Le problème du flowshop à deux machines avec des opérations couplées sur la première machine n'a pas été abordé dans la littérature. En se basant sur la classe des problèmes résolubles en des temps polynomiaux citée ci-dessus, nous avons donné une extension aux résultats trouvés, en considérant une deuxième machine pour former un problème du flowshop à deux machines. Cependant, nous avons déterminé plusieurs problèmes pour lesquels nous avons étudié leur complexité dans le reste de ce travail. La figure 3.3 regroupe l'ensemble des problèmes que nous avons déterminé et la relation existante entre eux.

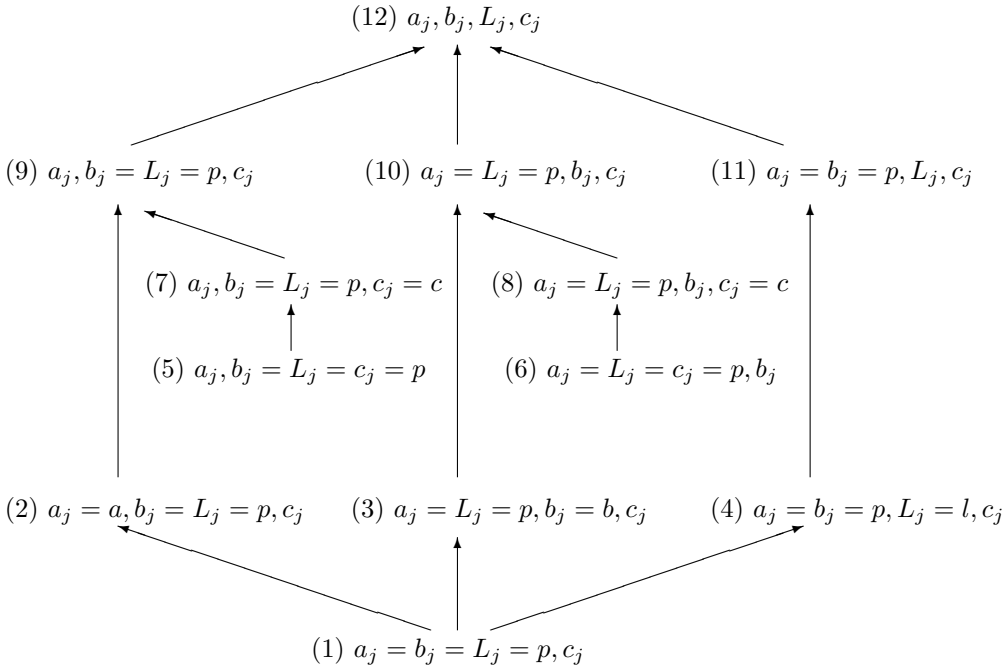

Figure 3.3 – Classification des problèmes

Nous terminons cette section par la description du problème général et nous donnons les notations utilisées comme suit. Nous considérons le problème d'ordonnancement du flowshop à deux machines M_1 et M_2. Soit J l'ensemble de n tâches à traiter sur les deux machines. Concernant le modèle 1, chaque tâche j est composée d'un couple d'opérations $O_{1,j}$ et d'une seule opération $O_{2,j}$ qui devront être exécutées sur les deux machines M_1 et M_2, respectivement, dans cet ordre pour toutes les tâches. Chaque couple d'opérations $O_{1,j}$ de la tâche j est décrit par le triplet (a_j, L_j, b_j), où a_j et b_j sont les durées de traitement de la première et de la deuxième opération, respectivement, tandis que L_j est le temps de latence exact qui s'écoule entre la date de fin de traitement de la première opération et de la date de début de traitement de la deuxième opération. L'opération $O_{2,j}$ est décrite par sa durée de traitement c_j. Pour chaque tâche j, le traitement de l'opération $O_{2,j}$ ne peut débuter seulement que si le traitement du couple d'opérations $O_{1,j}$ est achevé. L'objectif est de déterminer une séquence de tâches qui minimise le makespan C_{max}. Ce problème est noté par $F2/Coup\_Opr(1), a_j, L_j, b_j, c_j/C_{max}$. Ce qui est du modèle 2, chaque tâche j est composée d'une seule opération $O_{1,j}$ et d'un couple d'opérations $O_{2,j}$ qui s'exécutent sur les machines M_1 et M_2, respectivement. Chaque couple d'opérations $O_{2,j}$ est décrit par le triplet (c_j, H_j, d_j), et l'opération $O_{1,j}$ est définie par sa durée de traitement a_j. L'objectif est de minimiser le makespan. Le problème est noté par $F2/Coup\_Opr(2), a_j, c_j, H_j, d_j/C_{max}$.

3.3 Problèmes NP-difficiles

3.3.1 Le problème $F2/Coup\_Opr(1), a_j, b_j = L_j = p, c_j/C_{max}$

Dans cette section, nous considérons le problème $F2/Coup\_Opr(1), a_j, b_j = L_j = p, c_j/C_{max}$, que nous notons par $F2(a_j, b_j = L_j = p, c_j)$. Nous montrons que ce problème est NP-difficile en utilisant une réduction du problème de 2-partitions avec cardinalités identiques connu pour être NP-difficile [50].

Le problème de partition avec cardinalités identiques (PCI) est défini comme suivant : étant donné un ensemble $E = \{e_1, e_2, \ldots, e_{2n}\}$ de $2n$ entiers positifs tel que $\sum_{j=1}^{2n} e_j = 2B$ et B un nombre entier. Existe-t-il une partition de E en deux sous

ensembles disjoints E_1 et E_2 tels que $\sum_{j\in E_1} e_j = \sum_{j\in E_2} e_j = B$ et $|E_1| = |E_2| = n$? Ce problème est NP-difficile si chaque $e_j > 1, j = 1, \ldots, 2n$. Dans notre preuve, nous supposons que $e_j > 1, j = 1, \ldots, 2n$ (les résultats de ce problème sont publiés dans [82],[91],[84]).
Etant donné une instance quelconque du problème PCI, nous construisons une instance τ du problème $F2(a_j, b_j = L_j = p, c_j)$ avec un ensemble de $(4n + 2)$ tâches comme suit :

- $2n$ tâches de types U, notées par U_j ;
- n tâches identiques, notées par V ;
- $n + 1$ tâches identiques, notées par W ;
- Une seule tâche notée T ;

Pour toutes les tâches, on pose $b_j = L_j = p$, $j = 1, \ldots, 4n + 2$, avec $p > B$. Les valeurs des durées opératoires des tâches sont données dans la table 3.1.

Tâches	a_j	c_j
$U_j, j = 1, \ldots, 2n$	$p - e_j$	e_j
V	$p + 1$	$4p$
W	p	0
T	$B + p + 1 - n$	$(4n + 1)p - 2B$

Table 3.1 – Durées opératoires des tâches

Nous montrons que le problème PCI a une solution si, et seulement si, le problème d'ordonnancement admet une solution S avec un $C_{max}(S) \leq y$, où $y = 4(2n + 1)p + 1$.

Dans ce qui suit, la notation $(VW)-$tâche représente la tâche composée (VW) telles que les tâches V et W sont entrelacées, et la première opération de V est ordonnancée à la première position. En outre, les tâches composées (VW) sont des tâches de durées de traitement $(4p + 1)$ et $4p$ sur les machines M_1 et M_2, respectivement, et un temps de latence $l = -p$ (voir la figure 3.4).

Dans ce qui suit, nous montrons que si le problème PCI a une solution alors le

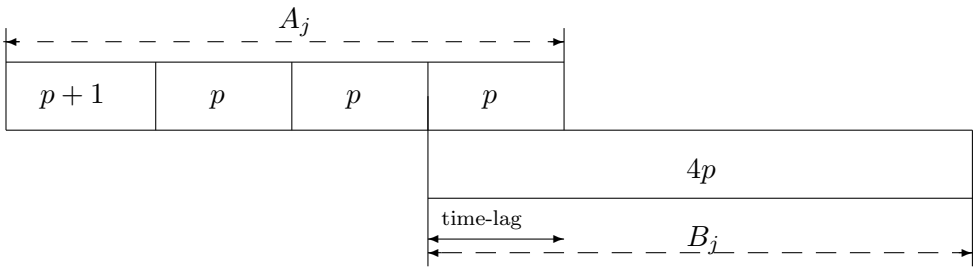

Figure 3.4 – Exemple d'une tâche composée (VW)

problème d'ordonnancement S admet une solution réalisable avec $C_{max}(S) \leq y$.

Lemme. 1 *Si le problème PCI a une solution alors l'instance τ du problème $F2(a_j, b_j = L_j = p, c_j)$ admet une solution réalisable S avec $C_{max}(S) \leq y$.*

Preuve. Soient E_1 et E_2 les deux partitions de l'ensemble E, tel que $\sum_{j \in E_1} e_j = \sum_{j \in E_2} e_j = B$, où les ensembles E_1 et E_2 contient chacun exactement n éléments. Nous pouvons ordonnancer les tâches correspondantes à l'instance τ comme suit.
Soient J_1 et J_2 les deux sous ensembles de $U-$tâches correspondants aux ensembles E_1 et E_2, respectivement. Soit S un ordonnancement réalisable de l'instance τ dans lequel les $U-$tâches de $J_1(J_2)$ sont entrelacées avec les $V-$tâches ($W-$tâches), et une tâche T est entrelacée avec la tâche W. Soient $(VU)-$tâches $\{(VU)_1, \ldots, (VU)_n\}$, $(WU)-$tâches $\{(WU)_1, \ldots, (WU)_n\}$ et $(WT)-$tâche, les tâches composées obtenues en entrelaçant les opérations. L'ordonnancement S est construit comme suit (voir figure 3.5) : nous commençons par ordonnancer $(VU)-$tâches, les tâches sont suivies de $(TW)-$tâche et nous terminons par $(WU)-$ tâches. L'ordre des tâches composées des ensembles $(VU)-$tâches, et $(WU)-$tâches dans l'ordonnancement S peut être choisi arbitrairement. Dans cet ordonnancement, il n'existe pas de temps mort entre les tâches composées sur les deux machines M_1 et M_2, donc,

$$C_{max} = p + 1 + 2p + 4np + \sum_{j \in J_1} e_j + (4n+1)p - 2B + \sum_{j \in J_2} e_j = 4p + 8np + 1$$

$$= 4(2n+1)p + 1. \square$$

Dans ce qui suit, nous montrons que s'il existe une solution pour le problème

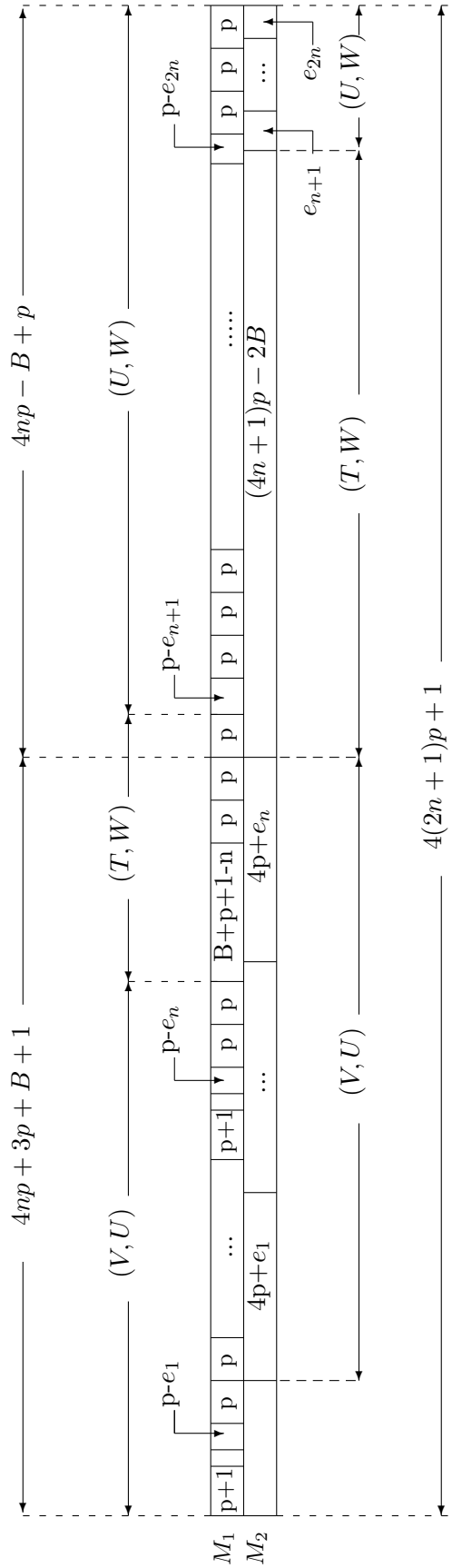

Figure 3.5 – Ordonnancement S

d'ordonnancement S avec un makespan inférieur ou égale à y, alors il existe une solution pour le problème PCI. Pour prouver ce résultat, nous montrons, dans un ordonnancement S, que :

1. Toutes les tâches sont entrelacées.
2. Il n'existe pas de tâches composées $(UU)-$tâches.
3. La $T-$tâche est entrelacée avec la $W-$tâche.

Soient X_1, X_2, Y_1, Y_2 les durées de traitement des tâches entrelacées données suivant la position de la tâche T dans l'ordonnancement S, comme le montre la figure 3.6. Soit $LB = X_1 + \max\{X_2, Y_2\}$ une borne inférieure pour le makespan de l'ordonnancement S.

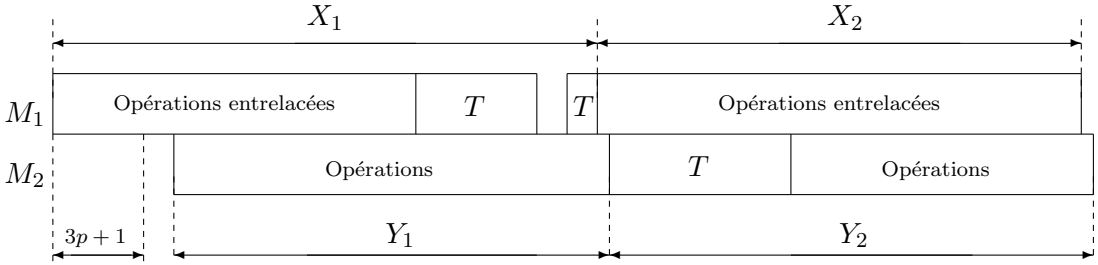

Figure 3.6 – Quelques notations

Soient A_j et B_j les durées de traitement des tâches composées sur les machines M_1 et M_2, respectivement, et soit L_j le temps de latence de la tâche composée comme le montre la figure 3.4. La table 3.2 résume les valeurs de A_j, B_j et L_j de toutes les tâches composées possibles de l'instance de notre problème d'ordonnancement.

Cependant, suivant les durées de traitement des tâches composées, nous avons :

- Dans la tâche composée (TU)-tâche, la tâche T ne peut être que dans la première position.
- Les tâches composées (U_jW) ont des durées de traitement inférieures à celles de (WU_j) sur la première machine.

Lemme. 2 *Dans un ordonnancement réalisable S, toutes les tâches sont entrelacées.*

Tâches composées	A_j	B_j	L_j
(VU_j)	$4p+1$	$4p+e_j$	$-p$
(VW)	$4p+1$	$4p$	$-p$
(TU_j)	$B+4p+1-n$	$(4n+1)p-2B+e_j$	$-p$
(TW)	$B+4p+1-n$	$(4n+1)p-2B$	$-p$
(U_iU_j)	$4p-e_i$	e_i+e_j	$-e_i$
(U_jW)	$4p-e_j$	e_j	$-e_j$
(WW)	$4p$	0	0

Table 3.2 – Durées opératoires des tâches composées

Preuve. Dans un ordonnancement S, supposons qu'il existe au moins deux tâches non entrelacées. Considérons que les $U-$tâches sont indexées dans l'ordre décroissant des valeurs de e_j de leurs durées de traitement A_j. Du moment que la durée totale de traitement des tâches composées sur la première machine est une borne inférieure du makespan, alors le meilleur entrelacement des opérations qui minimise la durée totale de traitement des tâches sur la première machine est une borne inférieure pour l'ordonnancement S.
Considérons l'entrelacement des opérations suivant : n tâches composées (VW), une tâche composée (TW), $(n-1)$ tâches composées (U_jU_k), où $j=n,\ldots,2n-2$; $k=1,\ldots,n-1$, et les tâches U_{2n-1} et U_{2n} sont ordonnancées seules (sans entrelacement), respectivement. Cet entrelacement d'opérations de tâches sur la première machine fournit une durée de traitement totale minimale lorsque exactement deux tâches ne sont pas entrelacées. Ainsi, en comparant la date de fin de traitement des tâches de l'ordonnancement S avec la somme des durées de traitement des tâches dans S sur la première machine, nous avons :

$$C_{max}(S) \geq (B+p+1-n+3p)+(4np+n)+4np+2p-\sum_{j=n-1}^{2n-2} e_j - e_{2n-1} - e_{2n}$$

$$\geq 4(2n+1)p+1+2p+B-\sum_{j=n-1}^{2n} e_j$$

$$\geq y+2p+B-\sum_{j=n-1}^{2n} e_j$$

Puisque $p > B$ et $\sum_{j=n-1}^{2n} e_j < 2B$, alors $2p + B - \sum_{j=n-1}^{2n} e_j > 0$. Donc, $C_{max}(S) > y$. Par conséquent, dans une solution réalisable, toutes les tâches sont entrelacées. □

Lemme. 3 *Dans un ordonnancement réalisable S, il n'existe pas de tâches composées $(UU)-$tâches.*

Preuve. Supposons qu'il existe une tâche composée $(UU)-$tâche dans un ordonnancement S. Supposons que cette $(UU)-$tâche est composée d'une pair (U_i, U_j) dans laquelle U_i est entrelacée avec U_j et les autres $U-$tâches sont entrelacées avec les autres types : $V-$tâches, $W-$tâches et $T-$tâche. Nous distinguons les cas suivants :

a. La tâche T est entrelacée avec une tâche de type $W-$tâches, donc il reste (n) $W-$tâches, (n) $V-$tâches et $(2n-2)$ $U-$tâches. Par conséquent, deux types d'entrelacement sont possibles, à savoir :

1. $(TW), (WW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n, |I_2| = n-2$ et $I_1 \cup I_2 \cup \{i,j\} = \{1,\ldots,2n\}$.
2. $(TW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-1, |I_2| = n-1$ et $I_1 \cup I_2 \cup \{i,j\} = \{1,\ldots,2n\}$.

b. La tâche T est entrelacée avec une $U-$tâche, alors il reste (n) $V-$tâches, $(n+1)$ $W-$tâches et $(2n-3)$ $U-$tâches. Par conséquent, trois types d'entrelacement sont possibles, à savoir :

1. $(TU_r), (WW), (WW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n, |I_2| = n-3$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.
2. $(TU_r), (WW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-1, |I_2| = n-2$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.
3. $(TU_r), (VW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-2, |I_2| = n-1$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.

Dans ce qui suit, nous vérifions la valeur du makespan de chaque séquence des tâches entrelacées.

Cas a.1 : D'après l'algorithme de Mitten [92], l'ordonnancement optimal des tâches composées de ce cas est donné par la séquence $S = \langle (VU_k)_{k\in I_1}, (TW), (U_iU_j), (U_kW)_{k\in I_2}, (WW)\rangle$. Suivant les durées de traitement des tâches composées données dans la table 3.2, les valeurs de X_1, X_2, Y_1 et Y_2 (voir figure 3.6) sont données, selon l'ordre des tâches composées de la séquence S, par :

$$X_1 = 4np + 3p + B + 1, \qquad X_2 = 4np + p - (\sum_{k\in I_2} e_k + e_i).$$
$$Y_1 = 4np + \sum_{k\in I_1} e_k, \qquad Y_2 = 4np + p - 2B + (\sum_{k\in I_2} e_k + e_i + e_j).$$

Donc, la borne inférieure de $C_{max}(S)$ est :

$$LB = X_1 + \max\{X_2, Y_2\}$$

$$LB = 8np + 4p + 1 + \max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j) - B\}$$

$$LB = y + \max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j) - B\}$$

Du moment que $\max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j) - B\} > 0$, alors $LB > y$. Ainsi, S n'est pas une solution réalisable avec $C_{max}(S) \leq y$.

Cas a.2 : D'après l'algorithme de Mitten [92], l'ordonnancement optimal des tâches composées de ce cas est donné par la séquence $S = \langle (VU_k)_{k\in I_1}, (TW), (VW), (U_iU_j), (U_kW)_{k\in I_2}\rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 sont :

$$X_1 = 4np - p + B, \qquad X_2 = 4np + 5p + 1 - (\sum_{k\in I_2} e_k + e_i).$$
$$Y_1 = 4np - 4p + \sum_{k\in I_1} e_k, \qquad Y_2 = 4np + 5p - 2B + (\sum_{k\in I_2} e_k + e_i + e_j).$$

La valeur de la borne inférieure est :

$LB = X_1 + \max\{X_2, Y_2\}$

$LB = 8np + 4p + 1 + \max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j) - B - 1\}$

$LB = y + \max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j) - B - 1\}.$

Puisque $e_k > 1, \forall k$, nous avons $\max\{B-(\sum_{k\in I_2} e_k+e_i), (\sum_{k\in I_2} e_k+e_i+e_j)-B-1\} > 0$. Donc, $LB > y$. Ainsi, S n'est pas une solution réalisable avec $C_{max}(S) \leq y$.

Cas b.1 : D'après l'algorithme de Mitten [92], l'ordonnancement optimal des tâches composées de ce cas est donné par la séquence : $S = \langle (VU_k)_{k\in I_1}, (TU_r), (U_iU_j), (U_kW)_{k\in I_2}, (WW), (WW)\rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 sont :

$X_1 = 4np + B + 3p + 1, \qquad X_2 = 4np + p - (\sum_{k\in I_2} e_k + e_i).$

$Y_1 = 4np + \sum_{k\in I_1} e_k, \qquad Y_2 = 4np + p - 2B + (\sum_{k\in I_2} e_k + e_i + e_j + e_r).$

Donc, la borne inférieure de $C_{max}(S)$ est :

$LB = X_1+\max\{X_2, Y_2\}$ $LB = y+\max\{B-(\sum_{l\in I_2} e_l+e_i), (\sum_{l\in I_2} e_l+e_i+e_j+e_k)-B\}.$

On a $\max\{B - (\sum_{l\in I_2} e_l + e_i), (\sum_{l\in I_2} e_l + e_i + e_j + e_k) - B\} > 0$, donc $LB > y$. Ainsi, S n'est pas une solution réalisable avec $C_{max}(S) \leq y$.

Cas b.2 : D'après l'algorithme de Mitten [92], l'ordonnancement optimal des tâches composées de ce cas est donné par la séquence : $S = \langle (VU_k)_{k\in I_1}, (TU_r), (VW), (U_iU_j), (U_kW)_{k\in I_2}, (WW)\rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 sont :

$X_1 = 4np - p + B, \qquad X_2 = 4np + 5p - (\sum_{k\in I_2} e_k + e_i).$

$Y_1 = 4np - 4p + (\sum_{k\in I_1} e_k), \qquad Y_2 = 4np + 5p - 2B + (\sum_{k\in I_2} e_k + e_i + e_r).$

La borne inférieure de $C_{max}(S)$ est :

$LB = X_1 + \max\{X_2, Y_2\}$

$LB = y + \max\{B - (\sum_{k\in I_2} e_k + e_i + 1), (\sum_{k\in I_2} e_k + e_i + e_j + e_r) - B - 1\}.$

Comme $e_k > 1, \forall k$, nous avons $\max\{B - (\sum_{k\in I_2} e_k + e_i + 1), (\sum_{k\in I_2} e_k + e_i + e_j + e_r) - B - 1 > 0\}$, alors $LB > y$. Ainsi, S n'est pas une solution réalisable avec $C_{max}(S) \leq y$.

Cas b.3 : D'après l'algorithme de Mitten [92], l'ordonnancement optimal des tâches composées est donné par la séquence : $\langle (VU_k)_{k\in I_1}, (TU_r), (VW), (VW), (U_iU_j), (U_kW)_{k\in I_2}\rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 sont :

$X_1 = 4np - 5p - 1 + B, \qquad X_2 = 4np + 9p + 2 - (\sum_{k\in I_2} e_k + e_i).$

$Y_1 = 4np - 8p + \sum_{k\in I_1} e_k, \qquad Y_2 = 4np + 9p - 2B + (\sum_{k\in I_2} e_k + e_i + e_j + e_r).$

La borne inférieure de $C_{max}(S)$ est :

$LB = X_1 + \max\{X_2, Y_2\}$

$LB = y + \max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j + e_r) - B - 1\}.$

Comme $e_k > 1, \forall k$, nous avons $\max\{B - (\sum_{k\in I_2} e_k + e_i), (\sum_{k\in I_2} e_k + e_i + e_j + e_r) - B - 1\} > 0$, alors $LB > y$. Ainsi, S n'est pas une solution réalisable avec $C_{max}(S) \leq y$. Notons que si nous avons plus d'une tâche composée $(UU)-$tâches alors X_2 croit et $LB > y$. □

Lemme. 4 *Dans un ordonnancement réalisable S, la T-tâche est entrelacée avec la W-tâche.*

Preuve. Supposons que dans un ordonnancement S, une tâche T est entrelacée avec une tâche U_r de type $U-$tâche. A partir des lemmes 2 et 3, les seules tâches composées possibles sont :

1. $(VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, (VW) où $|I_1| = n-1$, $|I_2| = n$ et $I_1 \cup I_2 \cup \{r\} = \{1, \ldots, 2n\}$.
2. $(VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, (WW) où $|I_1| = n$, $|I_2| = n-1$ et $I_1 \cup I_2 \cup \{r\} = \{1, \ldots, 2n\}$.

Pour les cas précédents, les séquences optimales sont :

- Case 1. $S = \langle (VU_k)_{k\in I_1}, (TU_r), (VW), (U_kW)_{k\in I_2} \rangle$
- Case 2. $S = \langle (VU_k)_{k\in I_1}, (TU_r), (U_kW)_{k\in I_2}, (WW) \rangle$

Similairement à la preuve du lemme 3, il est facile de montrer que pour chaque cas cité ci-dessus, $C_{max}(S) > y$. Alors, dans un ordonnancement S, la tâche T est entrelacée avec la tâche W. □

Lemme. 5 *Si l'instance τ admet une solution réalisable S avec $C_{max}(S) \leq y$, alors le problème PCI a une solution.*

Preuve. Soit S une solution réalisable tel que $C_{max}(S) \leq y$. A partir des lemmes 2, 3 et 4, l'unique couple d'opérations des tâches entrelacées dans l'ordonnancement S est $(VU_k)_{k\in I_1}$, (TW) et $(U_kW)_{k\in I_2}$ tel que $|I_1| = n, |I_2| = n$ et $I_1 \cup I_2 = \{1, \ldots, 2n\}$. La séquence optimale de ces tâches composées est : $S = \langle (VU_k)_{k\in I_1}, (TW), (U_kW)_{k\in I_2} \rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 de cette séquence sont :

$$X_1 = 4np + p + 1 + B, \qquad X_2 = 4np + p - \sum_{k\in I_2} e_k.$$

$$Y_1 = 4np + \sum_{k\in I_1} e_k, \qquad Y_2 = 4np + p - 2B + \sum_{k\in I_2} e_k.$$

La borne inférieure de $C_{max}(S)$ est :

$$LB = X_1 + \max\{X_2, Y_2\} = y + \max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\}.$$

Comme $C_{max}(S) \leq y$, alors $\max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\} = 0$. Ainsi, $\sum_{k\in I_2} e_k = B$ et $\sum_{k\in I_1} e_k = B$. Donc nous obtenons une solution pour le problème PCI. □

A partir des lemmes 1 et 5, nous avons le résultat suivant :

Théorème. 1 *Le problème* $F2(a_j, b_j = L_j = p, c_j)$ *est NP-difficile.*

3.3.2 Le problème $F2/Coup\_Opr(1), a_j = L_j = p, b_j, c_j/C_{max}$

Dans cette section, nous montrons que le problème $F2/Coup\_Opr(1), a_j = L_j = p, b_j, c_j/C_{max}$, noté par $F2(a_j = L_j = p, b_j, c_j)$, est NP-difficile en utilisant une réduction à partir du problème de 2-partitions avec cardinalités identiques (PCI) utilisée précédement (les résultats de ce problème sont publiés dans [82],[86],[87]).

A partir d'une instance quelconque du problème de 2-partitions avec cardinalités identiques, nous construisons une instance τ du problème $F2(a_j = L_j = p, b_j, c_j)$ avec un ensemble de $(4n + 4)$ tâches comme suit :

- $2n$ Tâches de type U, notée U_j ;
- n tâches identiques notées V ;
- $n + 2$ tâches identiques notées W ;
- Une seule tâche notée T ;
- Une seule tâche notée R ;

Pour toutes les tâches, nous posons $a_j = L_j = p, j = 1, \ldots, 4n + 4$ où $p > B$. Les durées de traitement des tâches sur les machines M_1 et M_2 sont données dans la table 3.3.

Tâches	b_j	c_j
$U_j, j = 1, \ldots, 2n$	$p - e_j$	e_j
V	$p + 1$	$5p + 1$
W	p	0
T	$(n + 2)p + B + 1$	$4p(n + 1) - 2B + 1$
R	$p + 1$	0

Table 3.3 – Durées opératoires des tâches

Nous montrons que le problème PCI a une solution si, et seulement si, l'instance τ admet une solution réalisable S avec un $C_{max}(S) \leq y$, où $y = 9(n+1)p + n + 2$. Nous avons le résultat suivant :

Lemme. 6 *Si le problème PCI a une solution alors il existe un ordonnancement S avec $C_{max}(S) \leq y$.*

Preuve. Supposons que le problème PCI a une solution, et soient E_1 et E_2 des sous ensembles de E tels que $\sum_{j \in E_1} e_j = \sum_{j \in E_2} e_j = B$ et $|E_1| = |E_2| = n$. Soient J_1 et J_2 les sous ensembles de $U-$tâches correspondants aux sous ensembles E_1 et E_2, respectivement. Donc, l'ordonnancement S existe lorsque la valeur de la date de fin de traitement $C_{max}(S)$ des tâches de S est égale à $9(n+1)p + n + 2$. La séquence des tâches composées est donnée par $S = \langle (U_k V)_{k \in J_1}, (WT), (WU_k)_{k \in J_2}, (WR) \rangle$. □

Supposons maintenant qu'il existe un ordonnancement S avec $C_{max}(S) \leq y$, alors nous montrons que le problème PCI a une solution. Afin de montrer ce résultat, nous avons établi les résultats suivants :

1. Toutes les tâches sont entrelacées.
2. Il n'existe pas de tâches composées $(UU)-$tâches.
3. La $T-$tâche est entrelacée avec la $W-$tâche.
4. La $R-$tâche est entrelacée avec $W-$tâche.

La table 3.4 résume les valeurs de A_j, B_j et L_j pour toutes les tâches composées possibles, en relation avec l'instance du problème d'ordonnancement.

Les résultats suivants traitent les cas $(1) - (4)$, cités précédement.

Lemme. 7 *Dans un ordonnancement réalisable S, toutes les tâches sont entrelacées.*

Preuve. La preuve est similaire à la preuve du lemme 2. □

Lemme. 8 *Dans un ordonnancement réalisable S, il n'existe pas de tâches composées de type $(UU)-$tâches.*

Preuve. La preuve est similaire à la preuve du lemme 3. □

Tâches composées	A_j	B_j	L_j
(U_jV)	$4p+1$	$5p+1+e_j$	$-e_j$
(WV)	$4p+1$	$5p+1$	0
(WU_j)	$4p-e_j$	e_j	0
(U_jR)	$4p+1$	e_j	$-e_j$
(WW)	$4p$	0	0
(WR)	$4p+1$	0	0
(U_iU_j)	$4p-e_i$	e_i+e_j	$-e_i$
(U_jT)	$B+(n+5)p+1$	$4p(n+1)-2B+e_j$	$-e_j$
(WT)	$B+(n+5)p+1$	$4p(n+1)-2B+1$	0

Table 3.4 – Durées opératoires des tâches composées

Lemme. 9 *Dans un ordonnancement réalisable S, la tâche T est entrelacée avec la tâche W.*

Preuve. La preuve est similaire à la preuve du lemme 4. □

Lemme. 10 *Dans un ordonnancement réalisable S, la tâche R est entrelacée avec la tâche W.*

Preuve. Dans un ordonnancement S Supposons que la tâche R est entrelacée avec la $U-$tâche. Soit U_r la $U-$tâche entrelacée avec la tâche R. A partir des lemmes $7, 8$ et 9, les seules tâches composées possibles sont :

1. $(WT), (U_rR), (U_kV)_{k\in I_1}, (WU_k)_{k\in I_2}$ et (WW) tel que $|I_1| = n, |I_2| = n-1$ et $I_1 \cup I_2 \cup \{r\} = \{1, \ldots, 2n\}$.
2. $(WT), (U_rR), (WV), (U_kV)_{k\in I_1}$, et $(WU_k)_{k\in I_2}$, où $|I_1| = n-1, |I_2| = n$ et $I_1 \cup I_2 \cup \{r\} = \{1, \ldots, 2n\}$.

Pour les cas précédents, 1 et 2, les séquences optimales sont :

- Case 1. $S = \langle (U_kV)_{k\in I_1}, (WT), (WU_k)_{k\in I_2}, (U_rR), (WW)\rangle$
- Case 2. $S = \langle (U_kV)_{k\in I_1}, (WV), (WT), (WU_k)_{k\in I_2}, (U_rR)\rangle$

Similairement à la preuve du lemme 3, il est facile de montrer pour les cas précédents que $C_{max}(S) > y$. Alors, dans un ordonnancement S, la tâche R est entrelacée avec la tâche W. □

Lemme. 11 *Si l'instance τ admet une solution réalisable S avec $C_{max}(S) \leq y$, alors le problème PCI a une solution.*

Preuve. A partir des lemmes $7, 8, 9$ et 10, les seules opérations entrelacées dans l'ordonnancement S sont : $(WT), (WR)$, $(U_kV)_{k\in I_1}, (WU_k)_{k\in I_2}$, où $|I_1| = n, |I_2| = n$ et $I_1 \cup I_2 = \{1, \ldots, 2n\}$. La séquence optimale pour ces tâches composées est $S = \langle (U_kV)_{k\in I_1}, (WT), (WU_k)_{k\in I_2}, (WR)\rangle$. Les valeurs de X_1, X_2 et Y_2 de cette séquence sont :

$$X_1 = 5np + 5p + B + n + 1, \qquad X_2 = 4np + 4p + 1 - \sum_{k\in I_2} e_k$$

$$Y_2 = 4np + 4p + 1 - 2B + \sum_{k\in I_2} e_k.$$

La borne inférieure de $C_{max}(S)$ est :

$$LB = X_1 + \max\{X_2, Y_2\} = y + \max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\}.$$

Comme $C_{max}(S) \leq y$, alors $\max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\} = 0$. Ainsi, $\sum_{k\in I_1} e_k = B$ et $\sum_{k\in I_2} e_k = B$. Donc, nous obtenons une solution pour le problème PCI. □

A partir des lemmes 6 et 11, nous avons le résultat suivant.

Théorème. 2 *Le problème $F2(a_j = L_j = p, b_j, c_j)$ est NP-difficile.*

3.4 Problèmes polynomiaux

3.4.1 Le problème $F2/Coup\_Opr(1), a_j = a, b_j = L_j = p, c_j/C_{max}$

Dans cette section, nous considérons le problème $F2(a_j = a, b_j = L_j = p, c_j)$, pour lequel nous distinguons deux cas. Si $a > p$, il est clair qu'il est impossible d'entrelacer les tâches, alors chaque tâche est ordonnancée seule. Dans ce qui suit, nous supposons que $a \leq p$. Les lemmes suivants sont utilisés pour proposer un

algorithme polynomial qui permet de résoudre le problème (les résultats de cette section sont publiés dans [82],[84], [91]).

Lemme. 12 *Dans un ordonnancement optimal, toutes les tâches sont entrelacées sauf une tâche si n est impair.*

Preuve. Soit S un ordonnancement optimal. Supposons qu'il existe dans S deux tâches J_i et J_j qui sont ordonnancées séparément, et supposons que la tâche J_i est ordonnancée avant J_j. Soit S' le nouvel ordonnancement construit à partir de S dans lequel la tâche J_j est entrelacée avec J_i pour former une nouvelle composante de tâches (J_iJ_j). Cette composante est traitée à la position J_i dans S. La durée de traitement totale de S' sur la machine M_1 est réduite de $p + a$ en respectant S, et le makespan de S' n'accroit pas. Cependant, S' est un ordonnancement optimal. En répétant cette opération, nous obtenons un ordonnancement dans lequel les tâches sont entrelacées sauf la dernière tâche si le nombre de tâches est impair. □

Lemme. 13 *Sur la deuxième machine, les tâches sont ordonnancées dans l'ordre décroissant des c_j.*

Preuve. Soit S un ordonnancement optimal qui ne satisfait pas le lemme 13. Soient J_i et J_j les deux premières tâches de S pour lesquelles $c_i \leq c_j$, et J_i est ordonnancée avant J_j. Considérons le nouvel ordonnancement S' dans lequel les positions des tâches J_i et J_j sont permutées. Toutes les tâches composées ont la même durée de traitement sur la première machine, alors la valeur de la durée totale de traitement des tâches de S' sur la machine M_1 ne change pas. De plus, la durée de traitement des tâches composées contenants J_i dans S' n'est pas supérieure à celle des tâches de S. Ainsi, le nouvel ordonnancement S' est aussi optimal. En répétant cette opération, nous obtenons un ordonnancement avec les propriétés considérées. □

L'algorithme polynomial pour le problème $F2(a_j = a, b_j = L_j = p, c_j)$ est donné ci-dessous :

Algorithme 1: algorithme 1

début

  si $a > p$ **alors**

    Appliquer l'algorithme de Johnson telles que les durées de traitement des tâches sur la première et la deuxième machine sont données par : $p_{1j} = a + 2p$ et $p_{2j} = c_j$, $j = 1, \ldots, n$, respectivement.

  sinon

    Indexer les tâches suivant l'ordre décroissant de leurs c_j, $j = 1, \ldots, n$.

    Construire les tâches composées $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, n/2$, si n est pair, ou $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, (n-1)/2$ et $T_{\frac{n-1}{2}+1} = (J_n)$ si n est impair.

    Ordonnancer les tâches composées T_j sur les deux machines, le plutôt possible, suivant l'ordre de leurs indices.

  fin

fin

Théorème. 3 *L'algorithme* 1 *fournit une solution optimale pour le problème* $F2(a_j = a, b_j = L_j = p, c_j)$ *en* $O(nlogn)$.

Preuve. Si $a > p$, les tâches ne peuvent pas être entrelacées, alors l'ordonnancement optimal est obtenu en appliquant l'algorithme de Johnson [61] sur la liste des tâches. Si $a \leq p$, alors suivant les lemmes 12 et 13, toutes les tâches sont entrelacées sauf la dernière tâche si n est impair, et les tâches sont ordonnancées dans l'ordre décroissant de leurs durées de traitement sur la deuxième machine. L'algorithme 1 fournit un ordonnancement en respectant les résultats des lemmes 12 et 13 et cette solution est donnée en $O(nlogn)$. □

Notons que l'algorithme 1 résout le même problème lorsque $a = p$, et nous obtenons le résultat suivant.

Théorème. 4 *L'algorithme* 1 *fournit une solution optimale pour le problème* $F2(a_j = b_j = L_j = p, c_j)$ *en* $O(nlogn)$.

3.4.2 Le problème $F2/Coup\_Opr(1), a_j = L_j = p, b_j = b, c_j/C_{max}$

Dans cette section, nous considérons le problème $F2(a_j = L_j = p, b_j = b, c_j)$. Si $b > p$, alors il n'est pas possible d'entrelacer les tâches, sinon nous fournissons les résultats suivants (les résultats de ce problème sont publiés dans [82],[86],[87]).

Lemme. 14 *Dans un ordonnancement optimal, toutes les tâches sont entrelacées sauf une tâche si n est impair.*

Preuve. La preuve est similaire à la preuve du lemme 12. □

Lemme. 15 *Sur la deuxième machine, les tâches sont ordonnancées dans l'ordre décroissant des c_j.*

Preuve. La preuve est similaire à la preuve du lemme 13. □

Pour résoudre le problème $F2(a_j = L_j = p, b_j = b, c_j)$, nous proposons l'algorithme 2 détaillé ci-dessous.

Algorithme 2: algorithme 2

début

  si $b > p$ **alors**

    Appliquer l'algorithme de Johnson en considérant les durées de traitement des tâches sur la première et la deuxième machine comme suit : $p_{1j} = 2p + b$ et $p_{2j} = c_j$, $j = 1, \ldots, n$, respectivement.

  sinon

    Indexer les tâches dans l'ordre décroissant de leur c_j, $j = 1, \ldots, n$.

    Construire les tâches composées $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, n/2$, si n est pair, ou $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, (n-1)/2$ et $T_{\frac{n-1}{2}+1} = (J_n)$ si n est impair.

    Ordonnancer les tâches composées T_j sur les deux machines, le plutôt possible, suivant l'ordre de leurs indices.

  fin

fin

Théorème. 5 *L'algorithme* 2 *fournit une solution optimale pour le problème* $F2(a_j = L_j = p, b_j = b, c_j)$ *en* $O(nlogn)$.

3.4.3 Le problème $F2/Coup\_Opr(1), a_j = b_j = p, L_j = L, c_j/C_{max}$

Dans cette section, nous considérons le problème $F2(a_j = b_j = p, L_j = L, c_j)$ (les résultats de ce problème sont publiés dans [82],[86],[87]). Si $L < p$, alors il n'est pas possible d'entrelacer les tâches, dans le cas contraire, nous avons les résultats suivants.

Lemme. 16 *Dans un ordonnancement optimal, toutes les tâches sont entrelacées sauf une tâche si* n *est impair.*

Preuve. La preuve est similaire à la preuve du lemme 12. □

Lemme. 17 *Sur la deuxième machine, les tâches sont ordonnancées dans l'ordre décroissant des* c_j.

Preuve. La preuve est similaire à la preuve du lemme 13. □

L'algorithme polynomial pour le problème $F2(a_j = b_j = p, L_j = L, c_j)$ est détaillé dans l'algorithme 3.

Théorème. 6 *L'algorithme* 3 *fournit une solution optimale pour le problème* $F2(a_j = b_j = p, L_j = L, c_j)$ *en* $O(nlogn)$.

3.4.4 Le problème $F2/Coup\_Opr(1), a_j, b_j = L_j = p, c_j = c/C_{max}$

Dans cette section, nous considérons le problème $F2(a_j, b_j = L_j = p, c_j = c)$ (les résultats de ce problème sont publiés dans [82],[84]). Soient les ensembles de tâches $K_1 = \{J_j | a_j > p\}$ et $K_2 = \{J_j | a_j \leq p\}$, et soient n_1 et n_2 leurs tailles, respectivement. Nous supposons que les tâches de K_1 et K_2 sont indexées dans l'ordre croissant de leurs a_j. Il est clair que les tâches de K_1 ne peuvent être entrelacées qu'avec les tâches de K_2, telles que les tâches de K_1 sont traitées en première position

Algorithme 3: algorithme 3

début

  si $L < p$ **alors**

    Appliquer l'algorithme de Johnson en considérant les durées de traitement des tâches sur la première et la deuxième machine comme suit : $p_{1j} = 2p + L$ et $p_{2j} = c_j$, $j = 1, \ldots, n$, respectivement.

  sinon

    Indexer les tâches dans l'ordre décroissant de leur c_j, $j = 1, \ldots, n$.

    Construire les tâches composées $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, n/2$, si n est pair, ou $T_j = (J_{2j-1}J_{2j})$, $j = 1, \ldots, (n-1)/2$ et $T_{\frac{n-1}{2}+1} = (J_n)$ si n est impair.

    Ordonnancer les tâches composées T_j sur les deux machines, le plutôt possible, suivant l'ordre de leurs indices.

  fin

fin

dans les tâches composées. Cependant, une tâche de K_2 peut être entrelacée avec une tâche de K_1 ou avec une autre tâche de K_2.
Soit S_l l'ensemble des tâches dans lequel exactement l tâches de K_1 sont entrelacées avec l tâches de K_2, $0 \leq l \leq \min\{n_1, n_2\}$, et les autres tâches de K_1 ne sont pas entrelacées (ordonnancées seules). Notons également que le reste des tâches de K_2 sont entrelaceés entre elles ou non entrelacées. Les résultats suivants sont utilisés pour proposer un algorithme polynomial.

Lemme. 18 *Il existe un ordonnancement optimal de l'ensemble S_l tel que les l dernières tâches de K_2 sont entrelacées avec les l premières tâches de K_1.*

Preuve. Pour prouver ce résultat, nous montrons en premier que, dans un ordonnancement optimal S_l que les l dernières tâches de K_2 sont entrelacées avec les tâches de K_1, puis nous complétons la preuve en montrant que les tâches de K_2 sont entrelacées avec les l premières tâches de K_1.

(i) Soit S^0 un ordonnancement optimal de S_l qui ne satisfait pas la première hypothèse du lemme 18. Alors, il existe une tâche $J_i^{K_2}$, $i \leq n_2 - l$, qui est entrelacée avec une tâche de K_1. Du moment que l tâches de K_1 sont entrelacées avec les tâches de K_2, alors il existe une tâche $J_j^{K_2}$, $j \geq n_2 - l$, telle que la

tâche $J_j^{K_2}$ n'est pas entrelacée avec une tâche de K_1 dans l'ordonnancement S^0. Considérons un nouvel ordonnancement S^* de l'ensemble S_l obtenu à partir de l'ordonnancement S^0 dans lequel nous permutons les positions des tâches $J_i^{K_2}$ et $J_j^{K_2}$. Comme $a_j \geq a_i$ et $J_i^{K_2}$ et $J_j^{K_2}$ ont la même durée de traitement sur la deuxième machine, alors nous avons $C_{max}(S^*) \leq C_{max}(S^0)$. En répétant cette opération, nous vérifions la première hypothèse du lemme 18.

(ii) Soit S^0 un ordonnancement optimal de S_l qui satisfait la première hypothèse du lemme 18 et ne satisfait pas la seconde hypothèse. Alors, il existe deux tâches $J_i^{K_1}$ et $J_j^{K_1}$, $i < j, j > l$, telle que $J_j^{K_1}$ est entrelacée avec la tâche $J_k^{K_2}$ et $J_i^{K_1}$ est ordonnancée seule. Dans S^0, nous avons aussi $J_j^{K_1}$ est entrelacée avant $J_i^{K_1}$ ou bien elle est ordonnancée après $J_i^{K_1}$ suivant leurs durées de traitement. Dans ce qui suit, nous discutons les deux cas.

1. $J_i^{K_1}$ est ordonnancée avant $J_j^{K_1}$. Dans ce cas, nous distinguons trois cas dépendants de la contribution de $J_i^{K_1}$ et $J_j^{K_1}$ dans la valeur de $C_{max}(S^0)$: (a) $J_i^{K_1}$ et $J_j^{K_1}$ sont sur le chemin critique de la première machine, c.à.d. que les durées de traitement des tâches $J_i^{K_1}$ et $J_j^{K_1}$ sont comptabilisées dans le calcul de C_{max} sur la première machine. (b) $J_i^{K_1}$ et $J_j^{K_1}$ ne sont pas sur le chemin critique de la première machine, et (c) la tâche $J_i^{K_1}$ est sur le chemin critique de la première machine mais $J_j^{K_1}$ ne l'est pas. Considérons un nouvel ordonnancement S^* de l'ensemble S_l obtenu à partir de l'ordonnancement S^0 dans lequel $J_k^{K_2}$ forme une nouvelle composante de tâches avec $J_i^{K_1}$, et soit $C_{max}(S^*)$ le makespan de S^*. Si $C_{max}(S^0)$ est donné par le cas (a) ou (b), alors il est facile de vérifier que $C_{max}(S^*) = C_{max}(S^0)$. Si $C_{max}(S^0)$ est donné par le cas (c) donc, nous avons $c > p$ (voir figure 3.7).

   A partir de la figure 3.7, nous avons :

$$\begin{aligned} C_{max}(S^*) &\leq t + p + \max\{0, c - \delta - p\} + L - c \\ &\leq t + L + \max\{p - c, -\delta\} \\ &\leq t + L = C_{max}(S^0) \end{aligned}$$

2. $J_i^{K_1}$ est ordonnancée après $J_j^{K_1}$. De nouveau, nous distinguons trois situations a, b et c identiques au cas 1., dépendant de la contribution de

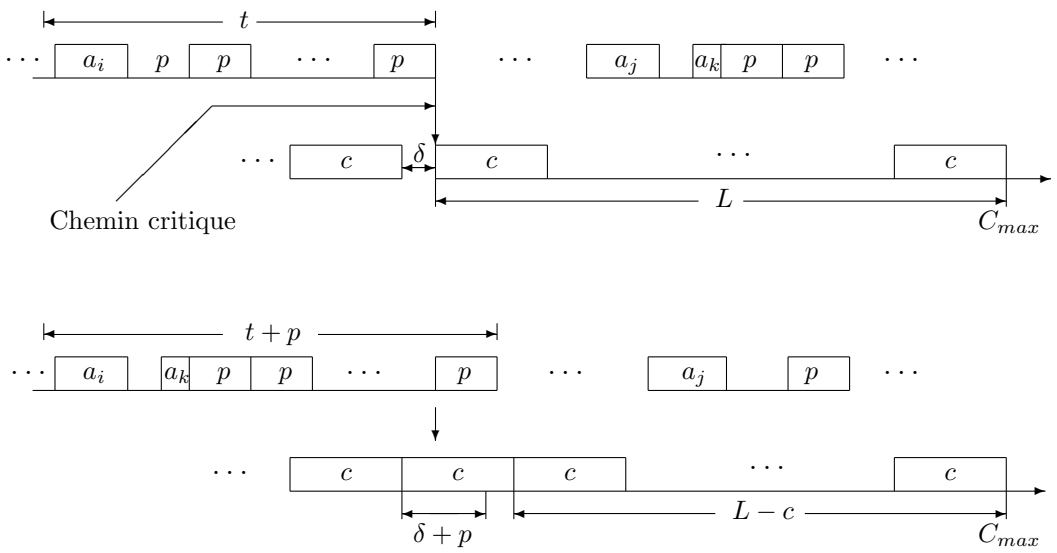

Figure 3.7 – La 3éme situation du cas 1 (avant et après le déplacement de $J_k^{k_2}$)

$J_i^{K_1}$ $J_j^{K_1}$ dans le calcul de la valeur de $C_{max}(S^0)$. La preuve des situations (a) et (b) sont similaires à celles du cas 1. Nous considérons seulement le cas dans lequel la tâche $J_j^{K_1}$ est sur le chemin critique de la première machine et la tâche $J_i^{K_1}$ ne figure pas sur le chemin critique. Supposons un nouvel ordonnancement S^* de l'ensemble S_l obtenu à partir de l'ordonnancement S^0 dans lequel les positions des tâches $J_i^{K_1}$ et $J_j^{K_1}$ sont permutées, et $J_k^{K_2}$ est entrelacée avec $J_i^{K_1}$. Soit $C_{max}(S^*)$ le makespan de S^*. A partir de la figure 3.8, nous avons $C_{max}(S^*) = t + L - \min\{\delta, a_j - a_i\} \leq C_{max}(S^0)$.

En répétant les opérations des cas 1 et 2, nous obtenons la seconde hypothèse du lemme 18. □

Dans ce qui suit, nous supposons que l'ensemble S_l satisfait le lemme 18.

Lemme. 19 *Il existe un ordonnancement optimal de l'ensemble S_l tels que :*

1. *le reste de toutes les tâches de K_2 sont entrelacées entre elles, sauf une tâche si leur nombre est impair ;*
2. *les tâches $J_1^{K_2}, \ldots, J_{\lceil \frac{n_2 - l}{2} \rceil}^{K_2}$ sont aux premières positions dans les nouvelles tâches composées.*

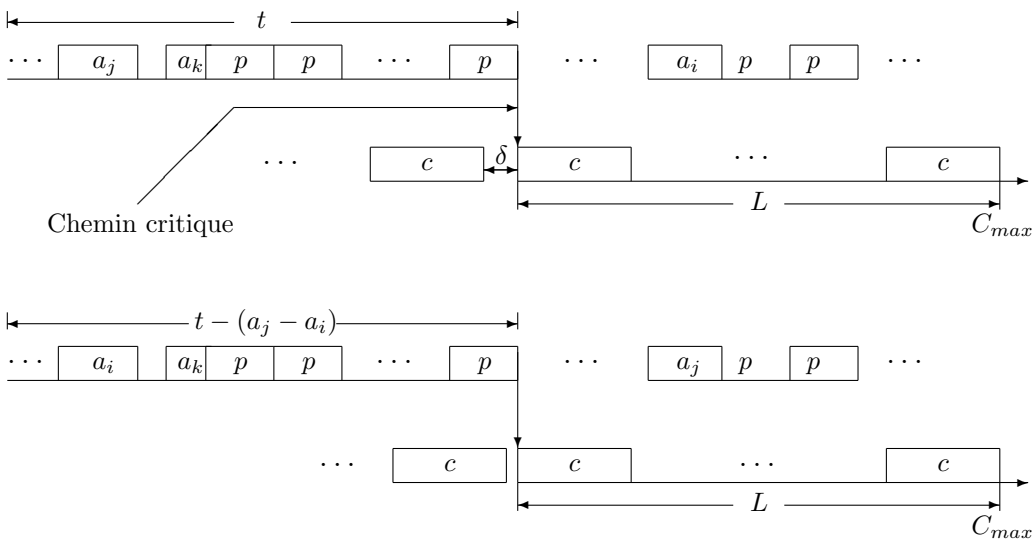

Figure 3.8 – La 3éme situation du cas 2 (avant et après la permutation de $J_i^{k_1}$ et $J_j^{k_1}$)

Preuve.

1. Soit S^0 un ordonnancement optimal de S_l qui ne satisfait pas le lemme 19. Soient J_i et J_j deux tâches de K_2 où $i < j$, et J_i et J_j sont traitées seules. D'après l'algorithme de Mitten, la tâche J_i est ordonnancée avant J_j. Suivant la contribution de J_i et J_j dans le calcul de la valeur de $C_{max}(S^0)$, nous distinguons trois cas : (a) J_i et J_j sont sur le chemin critique de la première machine, (b) J_i et J_j ne sont pas sur le chemin critique de la première machine, et (c) la tâche J_i est sur le chemin critique de la première machine mais pas la tâche J_j. Soit S^* le nouvel ordonnancement de l'ensemble S_l obtenu à partir de l'ordonnancement S^0 dans lequel la nouvelle tâche composée (J_iJ_j) est formée et ordonnancée à la position de J_i, et soit $C_{max}(S^*)$ son makespan. Si $C_{max}(S^0)$ est déterminé par les cas (a) ou (b), nous avons $C_{max}(S^*) \leq C_{max}(S^0)$. Donc S^* est optimal. Si le $C_{max}(S^0)$ est déterminé par le cas (c), nous obtenons $c < 3p$. Dans ce cas la valeur de $C_{max}(S^*)$ décroit de δ (voir figure 3.9), ce qui résulte que, $C_{max}(S^*) = t + (c - \delta) + (L - c) = C_{max}(S^0) - \delta$. En répétant cette opération, nous obtenons le résultat du lemme 19.
2. La deuxième hypothèse du lemme 19 peut être facilement démontrée en utilisant l'argument de permutation.

□

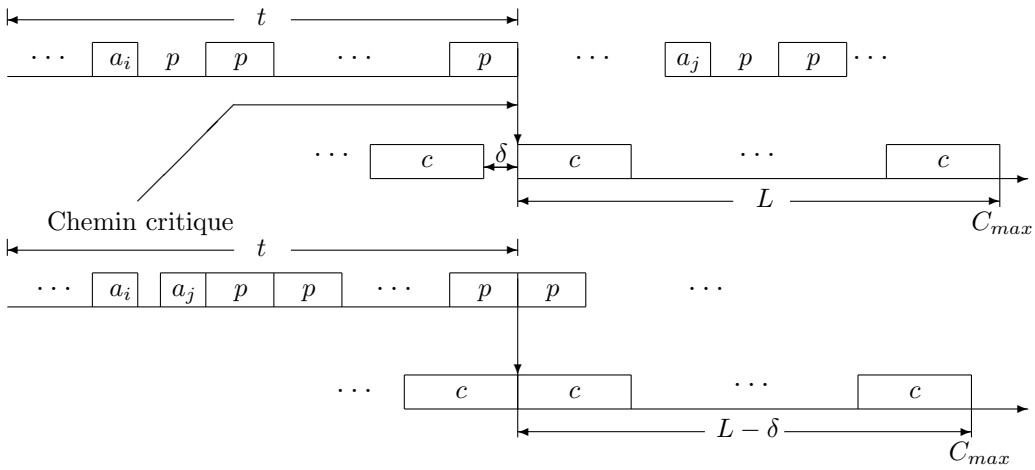

Figure 3.9 – La 3$^{\text{éme}}$ situation (avant et après le déplacement de la tâche J_j)

A partir des lemmes 18 et 19, nous avons le résultat suivant :

Lemme. 20 *L'algorithme de Mitten fournit un ordonnancement optimal de l'ensemble S_l en $O(n)$.*

Considérons l'algorithme suivant :

Algorithme 4: algorithme 4

début

Ranger les tâches de K_1 et K_2 dans l'ordre croissant de leurs a_j. Soit S^* un ordonnancement optimal. Poser $S^* = \emptyset$ et $C_{max}(S^*) = +\infty$.

pour $l := 0$ *à* $\min\{n_1, n_2\}$ **faire**

Entrelacer les l dernières tâches de K_2 avec les l premières tâches de K_1.

Entrelacer le reste des tâches de K_2 entre elles, telles que les tâches $J_1^{K_2}, \ldots, J_{\lceil \frac{n_2 - l}{2} \rceil}^{K_2}$ sont à la première position des nouvelles tâches composées.

Appliquer l'algorithme de Mitten pour ordonnancer les nouvelles tâches.

Soit S_l^* l'ordonnancement obtenu.

si $C_{max}(S_l^*) < C_{max}(S^*)$ **alors**

$S^* := S_l^*$ et $C_{max}(S^*) := C_{max}(S_l^*)$;

fin

fin

fin

Théorème. 7 *L'algorithme* 4 *fournit une solution optimale pour le problème* $F2(a_j, b_j = L_j = p, c_j = c)$ *en* $O(n^2 logn)$.

Preuve. Dans l'algorithme 4, l'ensemble construit à partir des tâches composées satisfait les lemmes 18 et 19 pour chaque valeur de l. De plus, l'algorithme 4 selectionne la meilleure solution parmi les valeurs de l. Alors, la solution optimale est obtenue pour le problème $F2(a_j, b_j = L_j = p, c_j = c)$, et l'algorithme 4 s'exécute en $O(n^2 logn)$. □

Notons que l'algorithme 4 résout le même problème lorsque $c = p$, d'où le résultat suivant.

Théorème. 8 *L'algorithme* 4 *fournit une solution optimale pour le problème* $F2(a_j, b_j = L_j = c_j = p)$ *en* $O(n^2 logn)$.

3.4.5 Le problème $F2/Coup\_Opr(1), a_j = L_j = p, b_j, c_j = c/C_{max}$

Dans cette section, nous considérons le problème $F2(a_j = L_j = p, b_j, c_j = c)$ (les résultats de ce problème sont publiés dans [82],[84]). Tout comme dans la section 3.4.4, nous décomposons l'ensemble de tâches en deux sous ensembles $K_1 = \{J_j | b_j > p\}$ et $K_2 = \{J_j | b_j \leq p\}$. Soient n_1 et n_2 les tailles des ensembles K_1 et K_2, respectivement. Nous supposons que les tâches de K_1 et K_2 sont indexées dans l'ordre croissant de leurs b_j. Les tâches de K_1 peuvent être entrelacées uniquement avec les tâches de K_2, où une tâche de K_1 est dans la deuxième position de la tâche composée. Cependant, une tâche de K_2 peut être entrelacée avec la tâche de K_1 ou bien avec une autre tâche de K_2.
Soit S_l l'ensemble de toutes les tâches dans lequel exactement l tâches de K_1 sont entrelacées avec l tâches de K_2, $0 \leq l \leq \min\{n_1, n_2\}$, et le reste des tâches de K_1 restent non entrelacées. Comme la durée de traitement de chaque tâche composée j sur la première machine est égale à $3p + b_j$, alors la tâche composée de l'ensemble S_l réalise la même fonction que la tâche composée de l'ensemble S_l de la section 3.4.4. D'où le résultat suivant.

Lemme. 21 *Il existe un ordonnancement optimal de l'ensemble* S_l *dans lequel les* l *dernières tâches de* K_2 *sont entrelacées avec les* l *premières tâches de* K_1.

Preuve. La preuve est similaire à la preuve du lemme 18. □

Lemme. 22 *Il existe un ordonnancement optimal de l'ensemble S_l tels que :*

1. *le reste de toutes les tâches de K_2 sont entrelacées entre elles, sauf une tâche si leur nombre est impair ;*
2. *les tâches $J_1^{K_2}, \ldots, J_{\lceil \frac{n_2-l}{2} \rceil}^{K_2}$ sont aux premières positions dans les nouvelles tâches composées.*

Preuve. La preuve est similaire à la preuve du lemme 19. □

Considérons l'algorithme 5 identique à l'algorithme 4 sauf que dans l'algorithme 5, les tâches de K_1 et K_2 sont rangées dans l'ordre croissant de leurs b_j.

Algorithme 5: algorithme 5

début
Ranger les tâches de K_1 et K_2 dans l'ordre croissant de leurs b_j. Soit S^* un ordonnancement optimal. Poser $S^* = \emptyset$ et $C_{max}(S^*) = +\infty$.
pour $l := 0$ *à* $\min\{n_1, n_2\}$ **faire**
Entrelacer les l dernières tâches de K_2 avec les l premières tâches de K_1.
Entrelacer le reste des tâches de K_2 entre elles, telles que les tâches $J_1^{K_2}, \ldots, J_{\lceil \frac{n_2-l}{2} \rceil}^{K_2}$ sont à la première position des nouvelles tâches composées.
Appliquer l'algorithme de Mitten pour ordonnancer les nouvelles tâches.
Soit S_l^* l'ordonnancement obtenu.
si $C_{max}(S_l^*) < C_{max}(S^*)$ **alors**
$S^* := S_l^*$ et $C_{max}(S^*) := C_{max}(S_l^*)$;
fin
fin
fin

Théorème. 9 *L'algorithme* 5 *fournit une solution optimale pour le problème $F2(a_j = L_j = p, b_j, c_j = c)$ en $O(n^2 logn)$.*

Preuve. La preuve est similaire à la preuve du théorème 7. □

Notons que l'algorithme 5 résout le problème lorsque $c = p$, d'où le résultat suivant :

Théorème. 10 *L'algorithme* 5 *fournit une solution optimale pour le problème $F2(a_j = L_j = c_j = p, b_j)$ en $O(n^2 logn)$.*

Le tableau suivant résume la complexité des problèmes étudiés du premier modèle.

Problème	Complexité	Référence
(1) $a_j = L_j = b_j = p, c_j$	$O(nlogn)$	Section 3.4.1
(2) $a_j = a, b_j = L_j = p, c_j$	$O(nlogn)$	Section 3.4.1
(3) $a_j = L_j = p, b_j = b, c_j$	$O(nlogn)$	Section 3.4.2
(4) $a_j = b_j = p, L_j = l, c_j$	$O(nlogn)$	Section 3.4.3
(5) $a_j, b_j = L_j = c_j = p$	$O(n^2logn)$	Section 3.4.4
(6) $a_j = L_j = c_j = p, b_j$	$O(n^2logn)$	Section 3.4.5
(7) $a_j, b_j = L_j = p, c_j = c$	$O(n^2logn)$	Section 3.4.4
(8) $a_j = L_j = p, b_j, c_j = c$	$O(n^2logn)$	Section 3.4.5
(9) $a_j, b_j = L_j = p, c_j$	NP-difficile	Section 3.3.1
(10) $a_j = L_j = p, b_j, c_j$	NP-difficile	Section 3.3.2
(11) $a_j = b_j = p, L_j, c_j$	NP-difficile	Orman et Potts [96]
(12) a_j, b_j, L_j, c_j	NP-difficile	Orman et Potts [96]

Table 3.5 – Complexité des problèmes du premier modèle

3.5 Deuxième modèle

Dans cette section, nous considérons le deuxième modèle du problème du flowshop à deux machines avec des tâches couplées sur la deuxième machine. Chaque tâche j possède deux opérations $O_{1,j}$ et $O_{2,j}$ à ordonnancer sur les machines M_1 et M_2, respectivement. $O_{1,j}$ est composée d'une seule opération avec une durée de traitement a_j, et $O_{2,j}$ est le couple d'opérations avec des durées de traitement et un délai exact donnés par le triplet (c_j, H_j, d_j). Pour chaque tâche j, le traitement du couple d'opération $O_{2,j}$ ne commence que si le traitement de l'opération $O_{1,j}$ est terminé. Comme notre objectif est de minimiser le makespan alors les deux modèles sont symétriques. En se basant sur les résultats trouvés dans les sections 3.3 et 3.4, nous déduisons les résultats de complexité des problèmes du modèle 2.

Problème	Complexité	Référence
$(1')$ $a_j, c_j = H_j = d_j = p$	$O(nlogn)$	Section 3.4.1
$(2')$ $a_j, c_j = H_j = p, d_j = d$	$O(nlogn)$	Section 3.4.1
$(3')$ $a_j, c_j = c, d_j = H_j = p$	$O(nlogn)$	Section 3.4.2
$(4')$ $a_j, c_j = d_j = p, H_j = h$	$O(nlogn)$	Section 3.4.3
$(5')$ $a_j = c_j = H_j = p, d_j$	$O(n^2logn)$	Section 3.4.4
$(6')$ $a_j = d_j = H_j = p, c_j$	$O(n^2logn)$	Section 3.4.5
$(7')$ $a_j = a, c_j = H_j = p, d_j$	$O(n^2logn)$	Section 3.4.4
$(8')$ $a_j = a, c_j, d_j = H_j = p$	$O(n^2logn)$	Section 3.4.5
$(9')$ $a_j, c_j = H_j = p, d_j$	NP-difficile	Section 3.3.1
$(10')$ $a_j, c_j, d_j = H_i = p$	NP-difficile	Section 3.3.2
$(11')$ $a_j, c_i = d_i = p, H_i$	NP-difficile	Orman et Potts [96]
$(12')$ a_j, c_j, H_j, d_j	NP-difficile	Orman et Potts [96]

Table 3.6 – Complexité des problèmes du second modèle

3.6 Conclusion

Dans ce chapitre, nous avons étudié la complexité du problème du flowshop à deux machines avec des opérations couplées pour minimiser la date de fin de traitement des tâches. Nous avons déterminé deux modèles. Dans le modèle 1, les opérations couplées sont considérées sur la première machine et dans le modèle 2, les opérations couplées sont considérées sur la deuxième machine. Nous avons étudié la complexité des problèmes du modèle 1. Nous avons prouvé la NP-complétude de certains problèmes et nous avons présenté des sous problèmes résolubles en des temps polynomiaux. Comme le critère à minimiser est le makespan, alors les deux modèles sont symétriques. Enfin, en utilisant les résultats du premier modèle, nous avons fourni des résultats de complexité des problèmes définis dans le second.

CHAPITRE 4

RÉSOLUTION ET EXPÉRIMENTATIONS NUMÉRIQUES

4.1 Introduction

Pour résoudre le problème général $F2/Coup - Opr(1), a_j, L_j, b_j, c_j/C_{max}$, nous avons développé deux types d'approches de résolution dans ce chapitre, les heuristiques et les métaheuristiques afin de fournir une solution approchée au problème comme nous avons proposé des bornes inférieures. Nous présentons des heuristiques basées sur l'utilisation des règles de priorité (SPT et LPT), et sur la règle de Johnson. Nous présentons également deux métaheuristiques, le recuit simulé et l'optimisation par essaim particulaire (OEP). D'autre part, nous proposons une méthode hybride à partir du recuit simulé et l'OEP. Enfin, nous avons réalisé des expérimentations numériques sur des instances générées aléatoirement pour évaluer la performance des méthodes élaborées.

4.2 Bornes inférieures

Dans ce qui suit, nous proposons des bornes inférieures pour le problème $F2/Coup-Opr(1), a_j, L_j, b_j, c_j/C_{max}$.

Proposition. 1 $LB_1 = \sum_{j=1}^{n}(a_j + b_j) + \min_{1\leq j\leq n}\{c_j\}$ *est une borne inférieure pour le makespan.*

Preuve. Toutes les tâches doivent être exécutées sur la première machine, alors la borne $\sum_{j=1}^{n}(a_j + b_j)$ est une borne inférieure de la durée totale de traitement des tâches sur la première machine. De plus, au moins une tâche doit être exécutée sur la deuxième machine, d'où la borne LB_1. □

Proposition. 2 $LB_2 = \min_{1\leq j\leq n}\{a_j + L_j + b_j\} + \sum_{j=1}^{n} c_j$ *est une borne inférieure pour le makespan.*

Preuve. $\sum_{j=1}^{n} c_j$ est la borne inférieure de la durée totale de traitement des tâches sur la deuxième machine. D'autre part, le traitement des tâches sur la deuxième machine ne peut commencer que si le traitement d'au moins d'une tâche est terminé sur la première machine. □

Proposition. 3 $LB_3 = \sum_{j=1}^{n} a_j + \min_{1\leq j\leq n}\{L_j\} + \min_{1\leq j\leq n}\{c_j\}$ *est une borne inférieure pour le makespan.*

Preuve. $\sum_{j=1}^{n} a_j + \min_{1\leq j\leq n}\{L_j\}$ est la borne inférieure du problème de tâches couplées à une seule machine avec des time lags exacts. Elle est déduite en relaxant le problème en posant $b_j = 0$. Par ailleurs, au moins une tâche doit être exécutée sur la deuxième machine. □

Proposition. 4 $LB_4 = \sum_{j=1}^{n} b_j + \min_{1\leq j\leq n}\{L_j\} + \min_{1\leq j\leq n}\{c_j\}$ *est une borne inférieure pour le makespan.*

Preuve. $\sum_{j=1}^{n} b_j + \min_{1\leq j\leq n}\{L_j\}$ est la borne inférieure du problème de tâches couplées à une seule machine avec des time lags exacts. Elle est déduite en relaxant le problème en posant $a_j = 0$. Par ailleurs, au moins une tâche doit être exécutée sur la deuxième machine. □

En conséquence, $LB = \max\{LB_1, LB_2, LB_3, LB_4\}$ est aussi une borne inférieure.

4.3 Approches de résolution

Pour résoudre les problèmes du flowshop avec ses différentes variantes, plusieurs méthodes de résolution ont été développées, telles que les méthodes exactes, les heuristiques et les métaheuristiques. Les méthodes exactes fournissent des solutions optimales pour des instances de petites tailles. Lorsque la taille du problème est grande, le temps d'exécution croit exponnentiellement. Cependant, les chercheurs font appel à d'autres techniques pour résoudre les problèmes et trouver des solutions réalisables dans un temps acceptable. Ainsi, plusieurs recherches se sont focalisées pour développer ces techniques et sont classées en deux classes. La première classe représente les méthodes constructives qui englobent des heuristiques constructives, tandis que la seconde classe représente les méthodes amélioratrices, parmi lesquelles nous citons les métaheuristiques.

4.3.1 Approches heuristiques

Dans cette section, nous proposons quelques heuristiques pour la résolution du problème NP-difficile $F2/Coup - Opr(1), a_j, L_j, b_j, c_j/C_{max}$. Ces algorithmes sont basés sur l'application des règles de priorité (LPT, SPT) et de la règle de Johnson [61] sur les durées de traitement (a_j, b_j, c_j), et sur les valeurs des délais (L_j). Pour toutes les heuristiques, la première étape consiste à ranger les tâches suivant l'ordre défini par les règles citées précédement, puis en deuxième étape, il faut utiliser une méthode sérielle ou parallèle. Le prinicipe de la méthode sérielle est le suivant : une tâche prioritaire est sélectionnée et ordonnancée le plutôt possible. Le principe de la méthode parallèle : à chaque instant t, pour lequel il existe une machine disponible, ordonnancer une tâche prioritaire. Les trois premières heuristiques s'exécutent en $O(nlogn)$ dans le cas de l'application de la sélection sérielle, et en $O(n^2)$ dans le cas de l'application de la sélection parallèle. L'heuristique 4 s'exécute en $O(n^2)$ (les résultats des tests de ces heuristiques sont publiés dans [81],[85]).

Heuristique 1 :

1. Ranger les tâches suivant la règle LPT (Longuest Processing Time) des a_j (respectivement L_j, b_j, c_j).
2. Utiliser la méthode en série (respectivement en parallèle) pour déterminer la

séquence des tâches à ordonnancer en les entrelaçant.

Heuristique 2 :

1. Ranger les tâches suivant la règle SPT (Shortest Processing Time) des a_j (respectivement L_j, b_j, c_j).
2. Utiliser la méthode en série (respectivement en parallèle) pour déterminer la séquence des tâches à ordonnancer en les entrelaçant.

Heuristique 3 :

1. Pour chaque tâche, poser : $p_{1j} = a_j + L_j + b_j$ et $p_{2j} = c_j$.
2. Ranger les tâches suivant la règle de Johnson.
3. Utiliser la méthode en série (respectivement en parallèle) pour déterminer la séquence des tâches à ordonnancer en les entrelaçant.

Heuristique 4 (H^*) :

1. Former deux listes de tâches $liste1$ et $liste2$ en rangeant les tâches suivant la régle LPT (Longuest Processing Time) des a_j et b_j, respectivement.
2. Pour chaque tâche de $liste1$, parcourir la liste des tâches $liste2$ et entrelacer la tâche de $liste1$ avec la première tâche non entrelacée de $liste2$.

4.3.2 Métaheuristiques

Actuellement, les métaheuristiques sont les approches les plus adaptées à un large nombre de problèmes d'optimisation combinatoire dans plusieurs domaines. Une caractéristique commune à de nombreuses métaheuristiques est qu'elles sont développées en s'inspirant des processus existants dans la nature, en physique ou en biologie. L'optimisation par essaims particulaires (OEP) est l'une des plus récentes métaheuristiques appliquées avec succès sur les problèmes d'optimsation combinatoire pour laquelle peu de travaux sont réalisés. Tasgetiren et al. [117] sont les premiers à développer un algorithme de l'OEP pour résoudre le problème d'ordonnancement à une seule machine pour minimiser la somme des retards pondérés,

comme ils ont proposé dans [118] une OEP pour deux problèmes, celui du flowshop de permutation pour minimiser le makespan et la grande tardiveté (L_{max}), respectivement. Les mêmes auteurs ont présenté dans [116] une OEP pour résoudre le problème du flowshop de permutation pour minimiser le makespan et la durée totale du flot. Toujours, pour le flowshop de permutation et dans le but de minimiser le makespan, Zhingang Lia et al. [128] ont proposé un algorithme de l'OEP pour résoudre le problème, et ils ont suggéré également, dans [129], une OEP pour le problème du job shop pour minimimser le makespan. Pour ce dernier problème, Sha et Cheng ont présenté une OEP hybridée [109]. Xianpeng et Linxin ont développé une OEP dans [124] pour le problème du flowshop avec la contrainte de blocage pour minimiser le makespan. Pour le flowshop sans attente, Quan et al. [100] ont introduit une OEP discrète dans le but de minimiser le makespan et la durée totale du flot. Dans [64], Karsan et Karimi ont adapté une OEP pour résoudre le problème de machines parallèles dans l'objectif de minimiser le makespan.

4.3.2.1 Optimisation par essaim de particules

L'optimisation par essaim de particules (OEP) est une méthode d'optimisation récente introduite par Kennedy et Eberhat [65]. Cette méthode s'inspire du comportement social de certains animaux évoluants en essaims tels que les bancs de poissons et les nuées d'oiseaux. En effet, il est étonnant de voir comment ces animaux se déplacent dans une seule direction, se divisant parfois en deux groupes pour éviter un prédateur puis reformant le groupe originel. Chaque individu utilise non seulement sa propre mémoire mais aussi l'information locale à laquelle il peut accéder sur le déplacement de ses plus proches voisins pour décider de son propre déplacement. De règles simples, telles que "rester relativement proche des autres individus", "aller dans la même direction", ou encore "aller à la même vitesse", suffisent à maintenir la cohésion du groupe et permettre des comportements collectifs complexes et adaptés.

Initialement, cette méthode a été conçue pour l'optimisation des fonctions non linéaires et continues. Elle repose sur une population appelée essaim, composée d'un ensemble d'individus disposés de façon aléatoire. Chaque individu est une particule représentant une solution potentielle pour le problème d'optimisation et soit f la fonction à minimiser. Soit N_p le nombre de particules ou la taille de la population.

Chaque particule est représentée par les caractéristiques suivantes :

1. Chaque particule k est représentée dans un espace de recherche de dimension n par son vecteur de position $X_k^t = (X_{k_1}^t, X_{k_2}^t, \ldots, X_{k_n}^t)$ et par son vecteur de vitesse $V_k^t = (V_{k_1}^t, V_{k_2}^t, \ldots, V_{k_n}^t)$ à l'itération t, où $X_{k_d}^t$: est la position de la particule k à la dimension $d = \overline{1,n}$, et $V_{k_d}^t$: la vitesse de la particule k à la dimension $d = \overline{1,n}$.
2. Chaque particule k mémorise sa meilleure position obtenue jusqu'à l'itération t, notée par $P_k^t = (P_{k_1}^t, P_{k_2}^t, \ldots, P_{k_n}^t)$.
3. $P_g^t = (P_{g_1}^t, P_{g_2}^t, \ldots, P_{g_n}^t)$ est la meilleure position de la meilleure particule dans l'essaim jusqu'à l'itération t.
4. La vitesse courante de la particule k de dimension d est obtenue en utilisant l'équation suivante :

$$V_{k_d}^t = wV_{k_d}^{t-1} + c_1 r_1 (P_{k_d}^{t-1} - X_{k_d}^{t-1}) + c_2 r_2 (P_{g_d}^{t-1} - X_{k_d}^{t-1}) \tag{4.1}$$

   w : coefficient d'inertie qui permet de contrôler l'effet de la vitesse précédente sur la vitesse présente de la particule.
   c_1, c_2 : deux constantes positives qui permettent de décider si les particules préfèrent se diriger dans la direction de sa meilleure position ou dans la direction de la meilleure position de ses voisins.
   r_1, r_2 : deux nombres aléatoires uniformément distribués sur $[0, 1]$;

5. Le vecteur de position de la particule k à l'itération t est donné par l'équation suivante :

$$X_{k_d}^t = X_{k_d}^{t-1} + V_{k_d}^t \tag{4.2}$$

6. La meilleure position de la particule k à l'itération t est donnée par :

$$P_k^t = \begin{cases} P_k^{t-1} & \text{si} \quad f(X_k^t) \geq f(P_k^{t-1}) \\ X_k^t & \text{si} \quad f(X_k^t) < f(P_k^{t-1}) \end{cases} \tag{4.3}$$

7. La meilleure position de la particule k dans la population est obtenue par :

$$P_g^t = \begin{cases} \text{argmin} \quad f(P_k^t) & \text{si} \quad \min f(P_k^t) < f(P_g^{t-1}) \\ P_g^{t-1} & \text{sinon} \end{cases} \tag{4.4}$$

L'utilisation de cette méthode ne se limite pas sur un espace de recherche continu, puisque beaucoup de problèmes d'optimisation sont définis sur des domaines à valeurs discrètes. Pour cela, des chercheurs ont développé plusieurs versions de l'algorithme discret de l'OEP qu'ils ont appliqué sur des problèmes d'optimisation discrète, en particulier dans les problèmes d'ateliers. Cependant, les auteurs ont proposé des algorithmes discrets de l'OEP pour certains problèmes cités dans [64],[65],[100].

a. Représentation de la solution :

L'une des clès dans la conception de l'OEP est la représentation ou le codage de sa solution, qui permet de trouver une solution dans une durée de temps acceptable. Dans le cas de notre méthode, chaque particule est représentée par un vecteur dont sa longueur est égale au nombre de tâches et chaque élément indique la position d'une tâche sur la machine.

tâche	T_1	T_2	T_3	T_4	T_5	T_6
position	4	1	6	5	3	2

Figure 4.1 – Codage d'une solution

b. Mise à jour de la particule :

Le comportement de la particule est un compromis entre trois choix possibles : trouver la position X_k^t de la particule, se diriger vers sa meilleure position P_k^t et trouver la meilleure position de la particule dans la population P_g^t. La position de la particule est adjustée par la formule suivante [100] :

$$X_k^t = c_2 \otimes F_3(c_1 \otimes F_2(w \otimes F_1(X_k^{t-1}), P_k^{t-1}), P_g^{t-1}) \tag{4.5}$$

Le calcul de la position de la particule consiste en trois composantes. La première composante correspond à la vitesse de la particule, appelée la composante physique du déplacement donnée par : $\lambda_k^t = w \otimes F_1(X_k^{t-1})$. L'opérateur F_1 représente l'opérateur de mutation avec la probabilité w tel que :

$$\lambda_k^t = \begin{cases} F_1(X_k^{t-1}) & \text{si } \ r < w \\ X_k^{t-1} & \text{sinon} \end{cases} \tag{4.6}$$

avec r un nombre réel qui suit la loi uniforme dans $[0, 1]$.

La deuxième composante est $\delta_k^t = c_1 \otimes F_2(\lambda_k^t, P_k^{t-1})$. Elle représente la composante cognitive du déplacement de la particule. Dans cette composante, F_2 indique l'opérateur de croisement avec la probabilité c_1. λ_k^t et P_k^{t-1} sont le premier et le deuxième parent du croisement, respectivement. Il en résulte que $\delta_k^t = F_2(\lambda_k^t, P_k^{t-1})$ ou bien $\delta_k^t = \lambda_k^t$, et cela selon le choix des valeurs du nombre aléatoire uniformément distribué.

La troisième composante est $X_k^t = c_2 \otimes F_3(\delta_k^t, P_g^{t-1})$. Elle représente la composante sociale du déplacement de la particule, où F_3 est l'opérateur du croisement avec la probabilité c_2. Notons que δ_k^t et P_g^{t-1} sont le premier et le second parent pour l'opérateur du croisement, respectivement. Alors, $X_k^t = F_3(\delta_k^t, P_g^{t-1})$ oubien $X_k^t = \delta_k^t$ selon le choix des valeurs du nombre aléatoire uniformément distribué. Le pseudocode de l'algorithme de l'OEP est donné dans l'algorithme 6.

Le critère d'arrêt peut être différent suivant le problème posé. Une erreur acceptable peut être définie comme un critère d'arrêt ou bien fixer un nombre maximum d'évaluations de la fonction objectif, sinon, fixer un nombre maximum d'itérations peut être aussi utilisé comme un critère d'arrêt.

Dans la méthode proposée ci-dessus, nous avons fait recours aux mécanismes de croisement et de mutation utilisés dans les algorithmes génétiques. Ces opérateurs permettent de diversifier suffisament la population au cours des générations, afin que l'algorithme ne reste pas bloqué dans un optimum local. L'opérateur de mutation est utilisé pour explorer différentes parties de l'espace de recherche, et l'opérateur de croisement tente d'exploiter le voisinage local pour trouver une solution qui améliore la fonction à optimiser (les résultats des tests de cette méthode sont publiés dans [81],[83],[89],[90]).

Algorithme 6: Optimisation par essaim de particules

Paramètres d'entrée : $t := 0$; N_p ;

pour $k := 1$ *à* N_p **faire**

  $X_k^t :=$ Générer aléatoirement une particule ;

  $P_k^t := X_k^t$;

fin

$P_g^t := X_l^t | l = \arg\min_{\forall k}\{C_{max}(X_k^t)\}$;

répéter

  pour $k := 1$ *à* N_p **faire**

    $V_k^t := wV_k^{t-1} + c_1 r_1 (P_k^{t-1} - X_k^{t-1}) + c_2 r_2 (P_g^{t-1} - X_k^{t-1})$;

    $X_k^t := X_k^{t-1} + V_k^t$;

    si $C_{max}(X_k^{t+1}) < C_{max}(P_k^t)$ **alors**

      $P_k^{t+1} := X_k^{t+1}$;

    sinon

      $P_k^{t+1} := P_k^t$;

    fin

  fin

  si $C_{max}(P_g^t) > \min_{1 \le l \le k}\{C_{max}(P_l^{t+1})\}$ **alors**

    $P_g^{t+1} := P_l^{t+1} | l = \arg\min_{\forall k}\{C_{max}(P_k^{t+1})\}$;

  fin

  t :=t+1 ;

jusqu'à;

critère d'arrêt est satisfait.

c. Mutation :

Dans la littérature, Il existe de nombreuses techniques de mutation employées dans les algorithmes génétiques. Dans ce travail, nous avons choisi la mutation par inversion que nous avons appliqué sur toutes les particules. Cet opérateur consiste à selectionner aléatoirement deux points de coupure sur un parent puis à inverser l'ordre des gènes situés au milieu, pour avoir une nouvelle séquence appelée enfant. La figure 4.2 illustre la manière dont nous procédons avec cet opérateur.

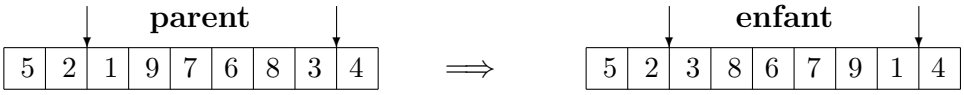

Figure 4.2 – Mutation par inversion

d. Croisement :

L'opérateur de croisement génère de nouvelles séquences en combinant deux autres séquences ou parents. Sur une longueur d'une séquence, deux points sont choisis aléatoirement pour déterminer un segment ou block. Ce block est recopié dans une nouvelle séquence appelée enfant, et le reste des gènes manquants du nouvel enfant sont complétés par ceux du deuxième parent en maintenant l'ordre. Une illustration du fonctionnement de l'opérateur de croisement utilisé est donnée dans la figure suivante.

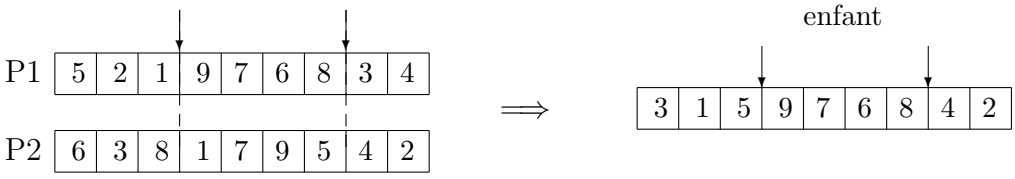

Figure 4.3 – Croisement par segment

4.3.2.2 Recuit simulé

Le recuit simulé a été introduit par Kirkpatrick et al. en 1983 [68]. Il est considéré comme une méthode de recherche locale simple utilisant une stratégie pour éviter les minimax locaux. Cette métaheuristique est inspirée d'un processus utilisé en métallurgie. La technique consiste à porter le matériau à une température élevée, où il devient liquide, puis à abaisser lentement celle-ci pour obtenir un état solide d'énergie minimale, qui correspond à une structure ordonnée et stable du solide.

Le recuit simulé est une méthode d'optimisation stochastique et itérative utilisée pour résoudre les problèmes d'optimisation combinatoire. Elle décrit l'évolution de l'équilibre thermodynamique d'un système dont la fonction objectif (f) à minimiser est son énergie (ce qui correspond au makespan, C_{max}, pour notre cas). A partir

d'une solution initiale "X_0", l'algorithme génére aléatoirement une nouvelle solution "X'" dans le voisinage de "X_0". Cette nouvelle solution est acceptée si la variation $\Delta f = f(X') - f(X_0) < 0$, et si $\Delta f \geq 0$, la solution X' est acceptée si la probabilité $p = \exp(-\Delta/T) > r$, où r un nombre généré aléatoirement suivant la loi uniforme dans $[0, 1]$, et T la température du système. La température est diminuée par paliers à chaque fois qu'un nombre d'itérations est effectué. La meilleure solution trouvée est mémorisée et l'algorithme est interrompu lorsque aucune solution voisine n'a été acceptée pendant un cycle complet d'itérations à température constante. L'algorithme du recuit simulé est présenté comme suit :

Algorithme 7: Recuit Simulé

Paramètres d'entrée : $T_0, T_f, T, \alpha, V_x, X_0, C_{max}$, Iteration_Paliers ;
$X := X_0$; $T := T_0$;
tant que $(T > T_f)$ **faire**
  Nbr_iter :=0 ;
  tant que *(Nbr_iter < iteration_paliers)* **faire**
    Choisir $X' \in V_x$;
    $\Delta C_{max} := C_{max}(X') - C_{max}(X)$;
    si $\Delta C_{max} < 0$ **alors**
      $X := X'$;
    sinon
      Tirer un nombre aléatoire dans $[0, 1]$;
      si $(r < \exp(-\Delta C_{max}/T))$ **alors**
        $X := X'$;
      fin
    fin
    Nbr_iter :=Nbr_iter+1 ;
  fin
  $T := \alpha * T$;
fin

La température initiale T_0 est maintenue constante pendant un certain nombre d'itérations ou paliers, puis elle diminue jusqu'à ce qu'elle soit suffisamment basse et elle passe à une température $T_k = \alpha^k * T_0$, où $\alpha \in [0, 1]$; un paramètre qui exprime la diminution de la température. Dans ce cas, le système est considéré comme gelé.

4.3.2.3 Hybridation de l'OEP avec le recuit simulé

L'optimisation par essaim particulaires a connu un grand succès dans beaucoup d'applications dans le domaine de l'optimisation, malgrès qu'elle est récente par rapport à certaines autres métaheuristiques. Cette approche de résolution a l'avantage de fournir une solution réalisable et de bonne qualité, mais son inconvénient majeur est la possibilité d'être bloquée dans des minimax locaux et possède une convergence lente. En outre, l'OEP est une méthode simple et facile à hybrider avec d'autres métaheuristiques ce qui a conduit à développer plusieurs variantes de son algorithme de base pour améliorer sa convergence et ses performances. D'autres part, le recuit simulé est un algorithme qui est connu par sa possibilité d'empêcher le problème des minimax locaux. Par conséquent, nous exposons dans ce qui suit l'algorithme hydride (Algorithme 8) que nous avons conçu en combinant les deux approches, l'optimisation par essaims particulaires et le recuit simulé (les résultats des tests de cette méthode sont publiés dans [81],[83]).

Algorithme 8: L'algorithme (OEP-RS) Hybridé

Etape 1 : Initialisation (population et paramètres).

Etape 2 : Evaluer la fonction objectif pour chaque particule, et déterminer P_k^t et P_g^t.

Etape 3 : Mettre à jour la vitesse et la position de chaque particule.

Etape 4 : Evaluer la fonction objectif pour chaque particule.

Etape 5 : Appliquer le recuit simulé pour chaque particule jusqu'à ce que le critère d'arrêt soit satisfait comme suit :

Etape 5.1 : Poser X' la solution initiale pour le recuit simulé, X^{t+1} et X^t les positions des particules à l'itération $(t+1)$ et t, respectivement. Soit $X' := X^{t+1}$ et $X := X^t$. Si $\Delta C_{max} = C_{max}(X') - C_{max}(X) < 0$, alors remplacer la position de la particule X par X', sinon générer un nombre aléatoire $r \in [0, 1]$ et la position de la particule X' est acceptée selon le critère suivant $\exp(-\Delta C_{max}/T > r)$.

Etape 5.2 : Mettre à jour la vitesse et la position de la particule de X' et calculer la valeur de la fonction objectif.

Etape 6 : Calculer P_k^t et P_g^t.

Etape 7 : Si le critère d'arrêt est satisfait, aller à l'étape 8 sinon, aller à l'étape 3.

Etape 8 : Retourner la meilleure solution.

4.4 Expérimentations numériques

Pour évaluer la performance des heuristiques proposées, des expérimentations numériques ont été réalisées. Les heuristiques sont testées avec les différentes valeurs des nombres de tâches dans l'ensemble $\{10, 30, 50, 100, 250, 500, 1000\}$. Les durées de traitement et les valeurs du délai pour les différentes instances sont générées aléatoirement suivant la loi uniforme. Ces durées prennent leurs valeurs dans les intervalles suivants : $a_j, L_j, b_j, c_j \in [1, 50]$; $a_j, L_j, b_j, c_j \in [1, 100]$; $a_j, L_j, b_j, c_j \in [50, 100]$; et $a_j, L_j, b_j \in [100, 150], c_j \in [1, 50]$. Les tableaux $(4.1 - 4.4)$ montrent les résultats obtenus à partir des expérimentations. Pour chaque valeur du nombre de tâches, nous avons généré 100 instances. Dans chaque tableau, la ligne "C_{max}"

représente le nombre de fois où la solution trouvée par une heuristique est meilleure par rapport aux autres heuristiques, la ligne "*opt*" indique le nombre de fois où le C_{max} est égale à la valeur de la borne inférieure, et la ligne "$\overline{time}$" indique la durée moyenne d'exécution de chaque heuristique. Les lignes "$\overline{dev}$" et "$mdev$" indiquent la déviation moyenne et la déviation maximale de chaque heuristique, respectivement. La déviation d'une heuristique est calculée par la formule $dev(H) = \frac{C_{max}(H)-LB}{LB}$.
Après la génération des instances, nous avons appliqué les heuristiques proposées. L'application de la première heuristique (heuristique 1) suivant l'ordre décroissant des valeurs de (a_j, L_j, b_j, c_j) des tâches (i.e. suivant la règle LPT), en considérant la selection sérielle (S) ou parallèle (P) des tâches, nous a induit à déceler plusieurs heuristiques que nous avons noté par : $LPTa_jS, LPTa_jP, LPTb_jS, LPTb_jP, LPTL_jS$, $LPTL_jP, LPTc_jS$, et $LPTc_jP$. De même, l'application de la deuxième heuristique (heuristique 2) suivant l'ordre croissant des valeurs de (a_j, L_j, b_j, c_j) des tâches (i.e. suivant la règle SPT), en considérant la sélection sérielle (S) ou parallèle (P) des tâches, nous a induit aussi à déceler plusieurs heuristiques que nous avons noté par : $SPTa_jS, SPTa_jP, SPT\ b_jS, SPTb_jP, SPTL_jS, SPTL_jP, SPTc_jS$, et $SPTc_jP$.
L'application de la troisième heuristique (heuristique 3) basée sur la règle Johnson (J) en considérant la sélection sérielle (S) ou parallèle (P) des tâches fournit deux autres heuristiques notées par JS et JP.

Les figures 4.4, 4.5, 4.6 et 4.7 représentent la variation de C_{max} en fonction du nombre de tâches pour les différentes valeurs de a_j, L_j, b_j et c_j. Pour $(a_j, L_j, b_j, c_j \in [1, 50])$, l'heuristique $SPTL_jP$ donne de meilleurs résultats de C_{max} par rapport aux autres heuristiques, pour un petit nombre de tâches ($n = 10$ ou 30). Pour $n = 50$, l'heuristique $LPTa_jP$ est meilleure que les autres et les valeurs de C_{max} croient avec la croissance du nombre de tâches. Pour $(a_j, L_j, b_j, c_j \in [1, 100])$ et $n = 10$, l'heuristique $SPTL_jP$ fournit de meilleurs résultats par rapport aux autres heuristiques, alors que pour $n = 30$ l'heuristic $LPTa_jP$ est meilleure. En comparant les heuristiques dans le cas où $(a_j, L_j, b_j, c_j \in [50, 100])$, alors l'heuristique $SPTL_jP$ est meilleure et les valeurs augmentent avec la croissance du nombre de tâches. L'heuristique $LPTa_jP$ devient dominante pour $(n = 500, 1000)$. Pour $a_j, L_j, b_j \in [100, 150]$ et $c_j \in [1, 50]$, la solution trouvée par l'heuristique $SPTL_jP$ est meilleure en comparant avec les autres solutions, pour atteindre la valeur maximale pour $n = 250$. Cette valeur décroit et permet à l'heuristique $LPTa_jP$ de prendre la

première position pour $n = 1000$. En comparant ces heuristiques entre elles dans certains cas des tests, certaines sont meilleures par rapport aux autres, pour des petites valeurs de nombre de tâches. Nous avons également comparé les résultats des heuristiques utilisant la sélection parallèle avec celles qui utilisent la sélection sérielle (exemple : comparer $LPTa_jS$ avec $LPTa_jP$), et nous avons constaté que les heuristique basées sur la sélection parallèle sont plus performantes que celles utilisant la sélection en série.

Les durées moyennes d'exécution des heuristiques croient avec la croissance du nombre de tâches. Ainsi, nous avons constaté que les heuristiques qui utilisent les algorithmes sérielles nécessitent une durée d'exécution importante par rapport aux heuristiques qui utilisent les algorithmes parallèles.

Toujours dans le cadre de la résolution du problème étudié, nous avons aussi proposé des métaheuristiques que nous avons testé. Ces approches sont : l'optimisation par essaim particulaires (OEP), le recuit simulé (RS) et l'optmisation par essaim particulaires hybridée avec le recuit simulé (OEP-RS). Pour les expérimentations numériques, nous avons suivi le même principe avec lequel nous avons réalisé les tests sur les heuristiques.

Pour les expérimentations numériques, il faut d'abord fixer et déterminer les valeurs de certains paramètres, qui sont utilisés dans les algorithmes de l'OEP et le recuit simulé.

Les paramètres de l'optimisation par essaim de particules sont : la taille de l'essaim ou la population fixée à 45, les probabilités $c_1 = c_2 = 0.7$, et le facteur d'inertie $w = 0.5$. Ce qui est du recuit simulé, ses paramètres sont : la température initiale $T_0 = 100$, la température finale, $T_f = 0.5$, et le coefficient de la température, $\alpha = 0.99$. Initialement, la solution initiale est générée aléatoirement et elle offre plus d'opportunités pour obtenir des solutions diversifiées que l'utilisation des heuristiques constructives qui limitent l'espace de recherche.

Les tableaux 4.5 et 4.6 résument les résultats obtenus des expérimentations numériques réalisées qui sont également représentés par des diagrammes. Les figures

$(4.8 - 4.11)$ expriment la variation de C_{max} en fonction du nombre de tâches. Les algorithmes de L'OEP et de l'OEP-RS fournissent de meilleurs résultats par rapport au recuit simulé (RS), pour tous les tests. Cependant, OEP-RS domine l'algorithme de l'OEP. Pour la durée moyenne d'exécution, l'algorithme du RS nécessite une durée d'exécution inférieure comparé avec la durée d'exécution de l'OEP et OEP-RS. Egalement, dans la plupart des cas, la durée d'exécution nécessaire pour l'algorithme de l'OEP est inférieure à celle de l'algorithme de l'OEP-RS. Nous avons aussi noté que les valeurs de la déviation moyenne obtenues par l'algorithme de l'OEP-RS sont meilleures par rapport à celles obtenues avec les deux autres algorithmes, pour toutes les valeurs de tâches. Ainsi, nous avons constaté que l'algorithme hybridé de l'optimisation par essaim particulaires et du recuit simulé est plus efficace que les algorithmes des autres approches de résolution.

	$a_j, L_j, b_j, c_j \in [1,50]$																		
	Lptc$_j$S	Lptc$_j$P	Lpta$_j$S	Lpta$_j$P	LptL_jS	LptL_jP	Lptb$_j$S	Lptb$_j$P	Sptc$_j$S	Sptc$_j$P	Spta$_j$S	Spta$_j$P	SptL_jS	SptL_jP	Sptb$_j$S	Sptb$_j$P	JS	JP	H^*
$n = 10$																			
C_{max}	1	3	4	7	2	0	3	6	6	4	2	0	0	24	15	2	0	10	11
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	0.2	0.1	0.4	0.2	0.4	0.1	0.7	0.1	0.5	0.1	1	0.2	645.8	0	0	0.2	1.2	0.1	0.4
$\overline{dev}$	0.2517	0.2776	0.2940	0.2423	0.2706	0.3129	0.3150	0.2415	0.3097	0.3046	0.2333	0.3334	0.2763	0.2068	0.2067	0.3163	0.2498	0.2420	0.2829
mdev	0.4313	0.5465	0.5954	0.4089	0.4886	0.4509	0.5246	0.4918	0.5250	0.6396	0.4701	0.5727	0.5171	0.4355	0.4176	0.5329	0.5250	0.4686	0.7449
$n = 30$																			
C_{max}	0	3	0	21	0	0	0	16	0	4	0	0	0	35	0	0	0	4	17
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	4.1	0.2	4	0.4	649.6	0.6	3.1	0.4	5	0.2	1294	0.2	4	1	3.7	0.1	4.2	0.1	0.9
$\overline{dev}$	0.2851	0.2055	0.3166	0.1663	0.2965	0.2866	0.3224	0.1742	0.3115	0.2303	0.2831	0.2694	0.2967	0.1653	0.2858	0.2637	0.2906	0.2050	0.2526
mdev	0.4192	0.3402	0.4379	0.3029	0.4644	0.4234	0.4448	0.3367	0.4695	0.4300	0.4234	0.3912	0.4463	0.3786	0.4142	0.4164	0.4363	0.3675	0.5279
$n = 50$																			
C_{max}	0	2	0	46	0	0	0	16	0	1	0	0	0	26	0	0	0	0	9
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	11	0.4	1302	0.4	1301	0.8	11.2	0.7	655.9	0.6	1302	1.1	655.96	0.8	10.8	0.3	11.7	0.7	1.1
$\overline{dev}$	0.3047	0.2024	0.3253	0.1459	0.3033	0.2827	0.3296	0.1583	0.3196	0.2145	0.3080	0.2598	0.3027	0.1579	0.3051	0.2485	0.3103	0.1977	0.2426
mdev	0.3815	0.2928	0.3977	0.1956	0.3988	0.3911	0.4166	0.2186	0.4261	0.3149	0.4192	0.3449	0.4380	0.2396	0.3844	0.3241	0.4224	0.2926	0.4988
$n = 100$																			
C_{max}	0	0	0	63	0	0	0	21	0	0	0	0	0	13	0	0	0	0	3
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	42.1	1.3	1334	1	5205	1	3916	1.5	2624	0.5	43.8	1	3269	1.4	3271	1.1	43.4	1.2	1.9
$\overline{dev}$	0.3069	0.1789	0.3177	0.1248	0.3123	0.2753	0.3226	0.1348	0.3162	0.1869	0.3148	0.2416	0.3096	0.1441	0.3121	0.2319	0.2107	0.1754	0.2278
mdev	0.3684	0.2581	0.3745	0.1809	0.3818	0.3364	0.4041	0.1715	0.3817	0.2560	0.3634	0.3138	0.3690	0.2314	0.3718	0.3086	0.3754	0.2268	0.4283
$n = 250$																			
C_{max}	0	0	0	93	0	0	0	5	0	0	0	0	0	1	0	0	0	0	1
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	339.9	3.63	26704	4.69	22238	5.01	24185	4.47	22882	3.96	26168	4.21	20349	4.28	24874	4.54	339.2	4.95	28.21
$\overline{dev}$	0.3093	0.1625	0.3239	0.1030	0.3155	0.2689	0.3236	0.1239	0.3131	0.1655	0.3195	0.2232	0.3130	0.1371	0.3211	0.2167	0.3132	0.1594	0.2156
mdev	0.3479	0.2021	0.3569	0.1385	0.3492	0.3185	0.3570	0.1595	0.3517	0.2060	0.3587	0.2623	0.3582	0.1902	0.3557	0.2590	0.3596	0.2049	0.3402
$n = 500$																			
C_{max}	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	1498	11.28	25551	11.2	29393	12.24	24315	11.89	27950	11.12	24898	12.2	20456	11.48	33273	11.96	1481	11.5	38.64
$\overline{dev}$	0.3109	0.1527	0.3236	0.0887	0.3224	0.2633	0.3243	0.1244	0.3151	0.1526	0.3197	0.2142	0.3214	0.1337	0.3251	0.2072	0.3109	0.1490	0.2061
mdev	0.3341	0.1904	0.3511	0.1058	0.3501	0.2941	0.3564	0.1455	0.3465	0.1923	0.3445	0.2474	0.3465	0.1877	0.3526	0.2401	0.3389	0.1915	0.2833
$n = 1000$																			
C_{max}	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	7748	39.96	40245	40.34	3062	42.95	37708	40.32	41409	36.38	32156	41.82	40187	41.21	34052	40.86	7808	38.02	234.38
$\overline{dev}$	0.3114	0.1494	0.3231	0.0798	0.3293	0.2603	0.3310	0.1295	0.3188	0.1445	0.3231	0.2072	0.3291	0.1343	0.3305	0.2018	0.3138	0.1442	0.2005
mdev	0.3281	0.1728	0.3457	0.0959	0.3515	0.2783	0.3583	0.1456	0.3374	0.1711	0.3442	0.2326	0.3559	0.1697	0.3552	0.2223	0.3294	0.1775	0.2507

Table 4.1 – Résultats des tests des heuristiques ($a_j, L_j, b_j, c_j \in [1,50]$)

	$a_j, L_j, b_j, c_j \in [1,100]$																		
	Lptc$_j$S	Lptc$_j$P	Lpta$_j$S	Lpta$_j$P	LptL_jS	LptL_jP	Lptb$_j$S	Lptb$_j$P	Sptc$_j$S	Sptc$_j$P	Spta$_j$S	Spta$_j$P	SptL_jS	SptL_jP	Sptb$_j$S	Sptb$_j$P	JS	JP	H^*
$n = 10$																			
C_{max}	5	11	4	3	2	0	2	1	6	7	5	0	7	18	5	0	0	13	11
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	0.62	0.16	0.76	0	0.79	0.31	0.64	0	0.3	0.15	0.93	0.16	0.63	0.15	0.31	0.16	0.63	0	0.16
$\overline{dev}$	0.2622	0.2497	0.3056	0.2633	0.2908	0.3641	0.3263	0.2746	0.3290	0.3215	0.2552	0.3270	0.3112	0.2554	0.2437	0.3489	0.2595	0.2630	0.3204
mdev	0.3918	0.4043	0.5954	0.4105	0.4785	0.5822	0.5345	0.4069	0.5086	0.5283	0.4165	0.6146	0.4650	0.4849	0.4230	0.5546	0.4581	0.4501	0.7988
$n = 30$																			
C_{max}	0	6	0	28	0	0	0	17	0	1	0	0	1	25	1	2	0	6	13
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	4.4	0.47	4.52	0.31	4.03	0.32	5.16	0.47	648	0.64	650.2	0.15	649.5	0.31	650.1	0.15	4.52	0.31	0.94
$\overline{dev}$	0.2965	0.2142	0.3236	0.1762	0.3085	0.2997	0.3327	0.1890	0.3175	0.2405	0.2970	0.2824	0.3017	0.1805	0.2956	0.2688	0.3023	0.2149	0.2589
mdev	0.4133	0.4434	0.5193	0.3366	0.5445	0.5449	0.4855	0.3696	0.4773	0.3923	0.4236	0.5218	0.5532	0.4146	0.4397	0.3738	0.4586	0.4323	0.6243
$n = 50$																			
C_{max}	0	2	0	34	0	0	0	17	0	0	0	0	0	29	0	0	0	1	17
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	9.91	0.61	655.4	1.11	11.32	0.63	10.82	0.3	655.8	0.3	1947	0.32	655.7	0.31	656.53	0.32	11.31	0.46	0.94
$\overline{dev}$	0.2956	0.1996	0.3228	0.1516	0.3015	0.2822	0.3301	0.1597	0.3156	0.2179	0.3069	0.2633	0.3015	0.1564	0.3059	0.2517	0.3048	0.2008	0.2444
mdev	0.4004	0.2984	0.4030	0.2315	0.3836	0.3915	0.4110	0.2341	0.4136	0.3123	0.3872	0.3556	0.4412	0.2633	0.4053	0.3199	0.4093	0.2753	0.5087
$n = 100$																			
C_{max}	0	0	0	54	0	0	0	20	0	0	0	0	0	21	0	0	0	0	5
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	40.2	0.79	3268	1.39	1976.6	1.09	3268	0.31	1331	0.78	4561	1.24	1975	1.51	1977	0.77	40.89	1.11	2.33
$\overline{dev}$	0.3071	0.1851	0.3218	0.1272	0.3121	0.2759	0.3295	0.1367	0.3171	0.1898	0.3148	0.2409	0.3077	0.1447	0.3170	0.2331	0.3112	0.1793	0.2284
mdev	0.3824	0.2725	0.3716	0.1659	0.3828	0.3404	0.4272	0.1838	0.3775	0.2480	0.3848	0.3281	0.3761	0.2198	0.4172	0.3108	0.3685	0.2541	0.3918
$n = 250$																			
C_{max}	0	0	0	86	0	0	0	12	0	0	0	0	0	2	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	158.5	3.6	7911	2.33	11780	3.63	12427	2.62	7903	2.16	11785	2.51	13013	2.98	6564	3.3	160.1	2.82	3.56
$\overline{dev}$	0.3124	0.1688	0.3268	0.1112	0.3174	0.2722	0.3266	0.1223	0.3164	0.1738	0.3239	0.2319	0.3125	0.1387	0.3211	0.2251	0.3142	0.1648	0.2233
mdev	0.3549	0.3134	0.3624	0.1400	0.3548	0.3034	0.3652	0.1397	0.3519	0.2152	0.3596	0.2719	0.3542	0.1883	0.3679	0.2738	0.3531	0.2084	0.3281
$n = 500$																			
C_{max}	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	1016	8.41	5522	8.1	7424	9.1	5518	8.1	14520	8.4	5522	9.1	7444	8.4	4833	8.6	1031	8.6	33.1
$\overline{dev}$	0.3129	0.1576	0.3278	0.0951	0.3210	0.2703	0.3259	0.1151	0.3150	0.1594	0.3251	0.2189	0.3186	0.1373	0.3271	0.2125	0.3164	0.1534	0.2117
mdev	0.3367	0.1842	0.3524	0.1120	0.3471	0.3008	0.3541	0.1334	0.3385	0.1903	0.3565	0.2419	0.3454	0.1732	0.3535	0.2389	0.3471	0.1860	0.2875
$n = 1000$																			
C_{max}	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	6247	33.4	37597	35.47	34053	37.79	37390	35.34	36667	34	34186	36.47	3418	32.45	41593	35.8	6237	32.61	106.8
$\overline{dev}$	0.3147	0.1526	0.3276	0.0857	0.3267	0.2669	0.3283	0.1192	0.3178	0.1507	0.3280	0.2112	0.3257	0.1364	0.3304	0.2078	0.3174	0.1479	0.2071
mdev	0.3315	0.1753	0.3423	0.0988	0.3464	0.2882	0.3480	0.1318	0.3365	0.1701	0.3424	0.2346	0.3470	0.1606	0.3538	0.2310	0.3435	0.1675	0.2659

Table 4.2 – Résultats des tests des heuristiques $a_j, L_j, b_j, c_j \in [1,100]$

	$a_j, L_j, b_j, c_j \in [50, 100]$																		
	Lptc$_j$S	Lptc$_j$P	Lpta$_j$S	Lpta$_j$P	LptL_jS	LptL_jP	Lptb$_j$S	Lptb$_j$P	Sptc$_j$S	Sptc$_j$P	Spta$_j$S	Spta$_j$P	SptL_jS	SptL_jP	Sptb$_j$S	Sptb$_j$P	JS	JP	H^*
$n = 10$																			
C_{max}	4	1	0	12	7	0	0	10	2	5	6	5	4	16	13	3	0	9	3
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	0.7	0	0.7	0.1	0.4	0.1	0.7	0.1	0.7	0.1	0.4	0.5	0.3	0.2	0.3	0.1	0.9	0	0.4
$\overline{dev}$	0.2012	0.1889	0.2224	0.1735	0.1454	0.2417	0.2182	0.1604	0.1988	0.2094	0.1694	0.2310	0.1936	0.1317	0.1643	0.2294	0.1942	0.1803	0.2195
mdev	0.3426	0.3246	0.4042	0.3316	0.2936	0.3631	0.3702	0.3316	0.3637	0.3609	0.3268	0.3714	0.3354	0.3590	0.3065	0.3734	0.3489	0.3503	0.3872
$n = 30$																			
C_{max}	0	4	0	14	0	0	0	15	0	2	0	0	0	52	0	0	0	8	5
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	5.77	0.63	3.28	0.31	649.3	0.79	652.1	0.15	3.59	0	3.46	0.77	651.46	0.46	4.21	0.16	5.25	0.32	0
$\overline{dev}$	0.2392	0.1364	0.2734	0.1094	0.2260	0.2238	0.2835	0.1114	0.2582	0.1495	0.2509	0.2024	0.2439	0.0785	0.2586	0.2004	0.2428	0.1353	0.1972
mdev	0.3654	0.2342	0.3724	0.2092	0.3244	0.2974	0.3930	0.2298	0.3695	0.2459	0.3396	0.2964	0.3410	0.1891	0.3520	0.2843	0.3608	0.2290	0.3338
$n = 50$																			
C_{max}	0	0	0	16	0	0	0	9	0	1	0	0	0	74	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	13.4	0.4	1949	0.7	12.4	0.8	656.6	0.7	685.3	0.5	1304	0.5	12.4	0.8	659.46	0.6	12.2	0.8	1
$\overline{dev}$	0.2556	0.1227	0.2768	0.0898	0.2355	0.2149	0.2890	0.0983	0.2627	0.1268	0.2695	0.1863	0.2446	0.0630	0.2726	0.1887	0.2627	0.1172	0.1853
mdev	0.3409	0.1762	0.3634	0.1641	0.2975	0.2635	0.3535	0.1686	0.3561	0.1859	0.3663	0.2524	0.3257	0.1232	0.3471	0.2559	0.3336	0.1844	0.2896
$n = 100$																			
C_{max}	0	0	0	18	0	0	0	7	0	0	0	0	0	73	0	0	0	2	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	57.2	1.45	5223	1.62	65.08	1.49	1351	1.44	3286	1.62	1352	1.56	3284	1.6	3931	1.5	56.82	1.53	2.91
$\overline{dev}$	0.2636	0.0995	0.2862	0.0728	0.2510	0.2105	0.2910	0.0793	0.2688	0.0989	0.2826	0.1737	0.2539	0.0554	0.2839	0.1727	0.2625	0.0996	0.1705
mdev	0.3098	0.1488	0.3472	0.1216	0.2992	0.2457	0.3499	0.1363	0.3287	0.1574	0.3309	0.2383	0.3026	0.0750	0.3501	0.2185	0.3125	0.1541	0.2466
$n = 250$																			
C_{max}	0	0	0	28	0	0	0	1	0	0	0	0	0	71	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	326.7	4.03	21641	4.28	26086	4.18	17008	4.06	22216	3.86	25512	4.09	19042	4.05	21642	4.17	330.8	4	27.62
$\overline{dev}$	0.2697	0.0763	0.2941	0.0540	0.2589	0.2039	0.2882	0.0695	0.2715	0.0763	0.2914	0.1637	0.2600	0.0475	0.2871	0.1544	0.2702	0.0771	0.1529
mdev	0.3002	0.0947	0.3299	0.0756	0.2908	0.2273	0.3223	0.0934	0.3048	0.1041	0.3253	0.1940	0.2901	0.0641	0.3227	0.1944	0.3105	0.1020	0.1933
$n = 500$																			
C_{max}	0	0	0	59	0	0	0	0	0	0	0	0	0	41	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	1042	8.28	6111	9.03	6156	9.51	6104	8.96	7491	7.26	5533	7.55	4189	7.87	7425	8.87	1047	9.61	12.55
$\overline{dev}$	0.2732	0.0639	0.2950	0.0449	0.2732	0.2026	0.2894	0.0675	0.2820	0.0634	0.2919	0.1529	0.2737	0.0463	0.2897	0.1512	0.2727	0.0643	0.1500
mdev	0.2979	0.0754	0.3237	0.0618	0.2978	0.2178	0.3141	0.0856	0.3024	0.0784	0.3178	0.1713	0.2963	0.0574	0.3164	0.1729	0.2962	0.0776	0.1844
$n = 1000$																			
C_{max}	0	0	0	93	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	4628	24.38	30512	25.52	26743	31.66	30413	27.75	25518	25.04	31085	27.85	21593	26.26	30375	26.6	4613	24.23	156.8
$\overline{dev}$	0.2748	0.0561	0.2987	0.0370	0.2846	0.2011	0.2945	0.0700	0.2889	0.0547	0.2974	0.1485	0.2849	0.0460	0.2953	0.1447	0.2745	0.0562	0.1437
mdev	0.2917	0.0676	0.3153	0.0465	0.2981	0.2157	0.3181	0.0877	0.3147	0.0636	0.3154	0.1737	0.3070	0.0545	0.3169	0.1576	0.2942	0.0660	0.1690

Table 4.3 – Résultats des tests des heuristiques ($a_j, L_j, b_j, c_j \in [50, 100]$)

	$a_j, L_j, b_j \in [100,150], c_j \in [1,50]$																	
	Lptc_jS	Lptc_jP	Lpta_jS	Lpta_jP	LptL_jS	LptL_jP	Lptb_jS	Lptb_jP	Sptc_jS	Sptc_jP	Spta_jS	Spta_jP	SptL_jS	SptL_jP	Sptb_jS	Sptb_jP	JS	JP	H^*
$n = 10$																			
C_{max}	4	6	0	13	3	0	4	5	3	3	4	9	4	21	3	4	0	3	11
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	0.62	0	0.77	0.15	0.31	0	1.11	0	0.47	0.16	0.16	0.15	0.47	0	0.79	0	0.62	0.32	0.45
$\overline{dev}$	0.2068	0.1910	0.2264	0.1589	0.1637	0.2354	0.2220	0.1519	0.2252	0.1827	0.2052	0.1977	0.2090	0.1285	0.2012	0.2084	0.2082	0.1736	0.2050
mdev	0.3924	0.3654	0.4121	0.3793	0.3146	0.3861	0.3754	0.3766	0.4113	0.3801	0.3505	0.3781	0.4137	0.3686	0.3688	0.3774	0.4057	0.3654	0.4956
$n = 30$																			
C_{max}	0	4	0	19	0	0	0	13	0	5	0	0	0	46	0	2	0	2	9
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	4.67	0	650.06	0.77	4.35	0.15	4.85	0.16	650.01	0.16	4.54	0.32	4.21	0.16	4.18	0.47	4.52	0.32	1.26
$\overline{dev}$	0.2419	0.1253	0.2658	0.0950	0.2230	0.2073	0.2712	0.1018	0.2520	0.1226	0.2599	0.1861	0.2354	0.0723	0.2523	0.1826	0.2384	0.1250	0.1817
mdev	0.3635	0.2315	0.3964	0.2532	0.3791	0.3257	0.4104	0.2537	0.4233	0.2359	0.3906	0.2915	0.3714	0.2310	0.4082	0.2905	0.3758	0.2298	0.3306
$n = 50$																			
C_{max}	0	3	0	13	0	0	0	15	0	0	0	0	0	72	0	0	0	1	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	10.64	0.31	11.22	0.77	654.86	0.47	2594	0.32	656.9	0.79	655.5	0.64	656.25	0.31	11.06	0.32	12.15	0.31	1.41
$\overline{dev}$	0.2512	0.1029	0.2781	0.0820	0.2309	0.2012	0.2806	0.0854	0.2575	0.1067	0.2707	0.1758	0.2390	0.0512	0.2710	0.1712	0.2515	0.1053	0.1684
mdev	0.3271	0.1711	0.4103	0.1510	0.3183	0.2933	0.3888	0.1503	0.3461	0.1853	0.3691	0.2425	0.3323	0.1264	0.3856	0.2703	0.3720	0.1660	0.2863
$n = 100$																			
C_{max}	0	0	0	12	0	0	0	4	0	0	0	0	0	83	0	0	0	1	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	42.32	1.25	3271	1.26	1333	0.91	4562	0.93	687.8	1.87	3271	1.1	4504	1.39	1335	1.08	43.06	1.99	21.07
$\overline{dev}$	0.2561	0.0818	0.2833	0.0611	0.2387	0.1911	0.2785	0.0682	0.2579	0.0831	0.2808	0.1616	0.2417	0.0362	0.2741	0.1537	0.2571	0.0798	0.1524
mdev	0.3367	0.1420	0.3487	0.1105	0.2857	0.2334	0.3298	0.1218	0.3270	0.1417	0.3450	0.2183	0.2922	0.0722	0.3274	0.2312	0.3316	0.1493	0.2320
$n = 250$																			
C_{max}	0	0	0	14	0	0	0	1	0	0	0	0	0	86	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	259.53	3.88	16400	3.59	18974	3.31	15754	5.03	17681	3.58	17528	1.7	15045	4.07	17690	2.82	261.4	2.34	65.02
$\overline{dev}$	0.2627	0.0593	0.2876	0.0450	0.2521	0.1887	0.2802	0.0582	0.2629	0.0592	0.2845	0.1459	0.2530	0.0291	0.2793	0.1411	0.2625	0.0594	0.1404
mdev	0.3107	0.0850	0.3559	0.0684	0.2947	0.2127	0.3145	0.0976	0.3068	0.0891	0.3393	0.1876	0.2891	0.0403	0.3136	0.1800	0.3030	0.0826	0.1959
$n = 500$																			
C_{max}	0	0	0	27	0	0	0	0	0	0	0	0	0	73	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	1841	13.78	36668	14.36	42403	15.89	38541	14.16	39232	13.35	41005	14.88	38532	13.8	39238	14.65	1881	13.89	102.81
$\overline{dev}$	0.2628	0.0458	0.2883	0.0349	0.2640	0.1857	0.2842	0.0575	0.2717	0.0445	0.2879	0.1401	0.2650	0.0277	0.2845	0.1349	0.3635	0.0454	0.1338
mdev	0.2920	0.0627	0.3170	0.0499	0.2879	0.2019	0.3141	0.0788	0.3124	0.0591	0.3238	0.1627	0.2874	0.0359	0.3121	0.1617	0.2909	0.0598	0.1604
$n = 1000$																			
C_{max}	0	0	0	57	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0
opt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{time}$	4738	25.25	34018	27.35	2687	30.06	35505	28.01	28519	25.18	34351	29.74	29440	26.7	33763	29.18	4762	26.59	76.23
$\overline{dev}$	0.2645	0.0373	0.2889	0.0273	0.2766	0.1858	0.2883	0.0571	0.2779	0.0373	0.2892	0.1351	0.2771	0.0273	0.2886	0.1317	0.2634	0.0386	0.1305
mdev	0.2853	0.0442	0.3096	0.0397	0.2942	0.2029	0.3083	0.0746	0.2996	0.0483	0.3063	0.1550	0.2943	0.0324	0.3076	0.1479	0.2825	0.0460	0.1534

Table 4.4 – Résultats des tests des heuristiques ($a_j, L_j, b_j \in [100,150], c_j \in [1,50]$)

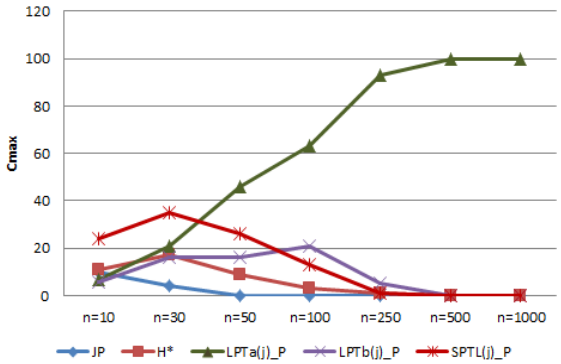

Figure 4.4 – $a_j, L_j, b_j, c_j \in [1, 50]$

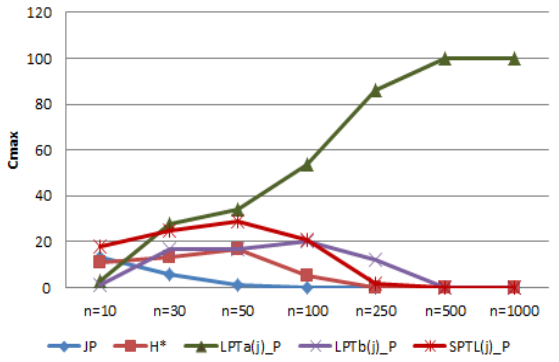

Figure 4.5 – $a_j, L_j, b_j, c_j \in [1, 100]$

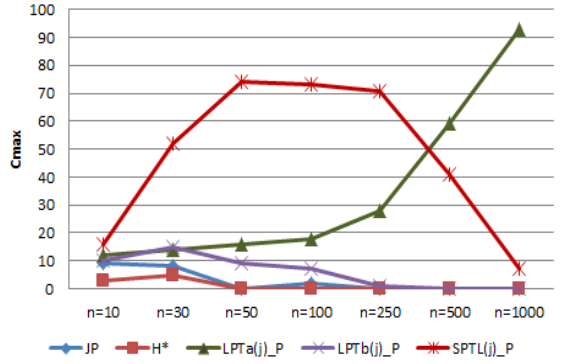

Figure 4.6 – $a_j, L_j, b_j, c_j \in [50, 100]$

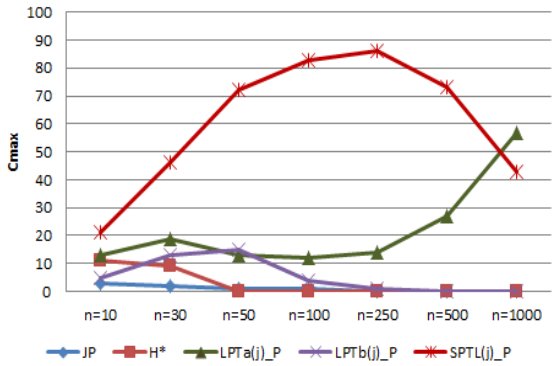

Figure 4.7 – $a_j, L_j, b_j \in [100, 150], c_j \in [1, 50]$

		$a_j, L_j, b_j, c_j \in [1, 50]$			$a_j, L_j, b_j, c_j \in [1, 100]$		
		RS	OEP	$OEP - RS$	RS	OEP	$OEP - RS$
n=10	C_{max}	0	30	70	1	35	64
	opt	0	0	0	0	0	0
	$\overline{time}$	2.81	186	400	1.60	98.2	318.4
	$\overline{dev}$	0.2117	0.1128	0.0892	0.1912	0.1286	0.1035
	mdev	0.5065	0.3710	0.3567	0.3737	0.3418	0.3247
n=30	C_{max}	0	14	86	0	15	85
	opt	0	0	0	0	0	0
	$\overline{time}$	5.7	384	2104	8.59	628.8	4361
	$\overline{dev}$	0.1899	0.1318	0.1174	0.1937	0.1294	0.1148
	mdev	0.3245	0.2695	0.2526	0.3237	0.2525	0.2295
n=50	C_{max}	0	10	90	0	7	93
	opt	0	0	0	0	0	0
	$\overline{time}$	10.51	886	4496	27.83	1485.8	9227.5
	$\overline{dev}$	0.1708	0.1334	0.1196	0.1833	0.1354	0.1219
	mdev	0.2334	0.1904	0.1791	0.2567	0.1841	0.1770
n=100	C_{max}	0	13	87	0	9	91
	opt	0	0	0	0	0	0
	$\overline{time}$	28.6	2240.5	14602	38.5	3336.9	24511
	$\overline{dev}$	0.1698	0.1408	0.1267	0.1702	0.1396	0.1315
	mdev	0.1736	0.1417	0.1318	0.2504	0.2060	0.1967
n=250	C_{max}	0	30	70	0	15	85
	opt	0	0	0	0	0	0
	$\overline{time}$	97.59	6842	27053	89.64	6148	25689
	$\overline{dev}$	0.1580	0.1375	0.1354	0.1610	0.1416	0.1384
	mdev	0.2029	0.1790	0.1746	0.2038	0.1802	0.1767
n=500	C_{max}	0	35	65	0	41	59
	opt	0	0	0	0	0	0
	$\overline{time}$	352.5	25694	33765	303	21668	38889
	$\overline{dev}$	0.1516	0.1363	0.1355	0.1561	0.1412	0.1401
	mdev	0.1865	0.1671	0.1663	0.1942	0.1721	0.1719

Table 4.5 – Résultats des tests des métaheuristiques

		$a_j, L_j, b_j, c_j \in [50, 100]$			$a_j, L_j, b_j \in [100, 150], c_j \in [1, 50]$		
		RS	OEP	$OEP - RS$	RS	OEP	$OEP - RS$
n=10	C_{max}	0	36	64	0	40	60
	opt	0	0	0	0	0	0
	$\overline{time}$	1.7	65.6	251.6	1.99	113.7	556.1
	$\overline{dev}$	0.1143	0.0698	0.0577	0.1124	0.0588	0.0543
	mdev	0.3598	0.3410	0.3412	0.3613	0.2945	0.2906
n=30	C_{max}	0	18	82	0	9	91
	opt	0	0	0	0	0	0
	$\overline{time}$	3.8	240.9	1138.7	5.32	387.3	2181
	$\overline{dev}$	0.1006	0.0583	0.0479	0.0933	0.0566	0.0467
	mdev	0.1935	0.1325	0.1261	0.2310	0.1842	0.1792
n=50	C_{max}	0	10	90	0	7	93
	opt	0	0	0	0	0	0
	$\overline{time}$	7.1	622	3788	12.14	921.8	5735
	$\overline{dev}$	0.1008	0.0590	0.0484	0.0834	0.0524	0.0423
	mdev	0.1734	0.1023	0.0844	0.1721	0.1302	0.1052
n=100	C_{max}	0	5	95	0	7	93
	opt	0	0	0	0	0	0
	$\overline{time}$	16.6	1334	9385	43.87	3109.3	24193
	$\overline{dev}$	0.0875	0.0575	0.0511	0.0734	0.0409	0.0366
	mdev	0.1328	0.0795	0.0668	0.1304	0.0809	0.0706
n=250	C_{max}	0	14	86	0	19	81
	opt	0	0	0	0	0	0
	$\overline{time}$	99.23	6977	30784	96.17	5739	30007
	$\overline{dev}$	0.0720	0.0525	0.0512	0.0494	0.0353	0.0341
	mdev	0.0993	0.0653	0.0624	0.0792	0.0581	0.0461
n=500	C_{max}	0	39	61	0	37	63
	opt	0	0	0	0	0	0
	$\overline{time}$	311.5	18918	42489	311.21	23627	45912
	$\overline{dev}$	0.0632	0.0495	0.0485	0.0462	0.0376	0.0312
	mdev	0.0776	0.0573	0.0561	0.0663	0.0424	0.0410

Table 4.6 – Résultats des tests des métaheuristiques

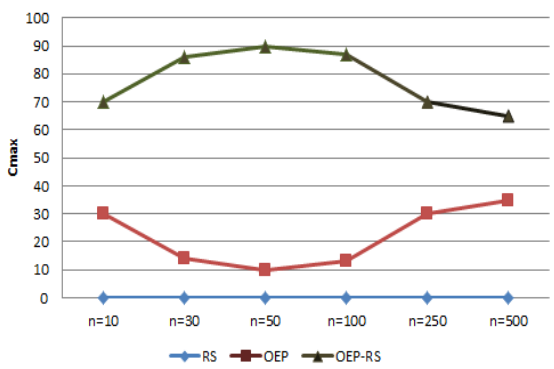

Figure 4.8 – $a_j, L_j, b_j, c_j \in [1, 50]$

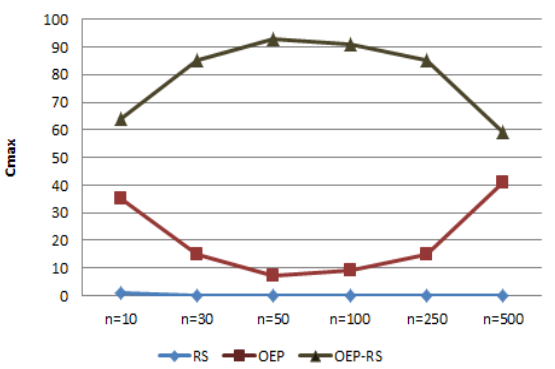

Figure 4.9 – $a_j, L_j, b_j, c_j \in [1, 100]$

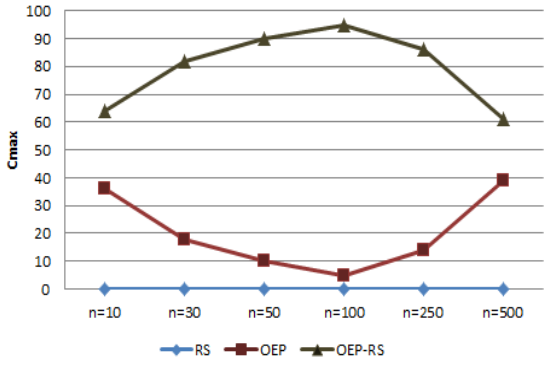

Figure 4.10 – $a_j, L_j, b_j, c_j \in [50, 100]$

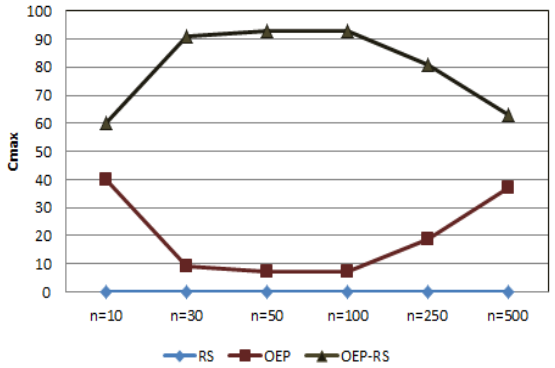

Figure 4.11 – $a_j, L_j, b_j \in [100, 150], c_j \in [1, 50]$

4.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la résolution du problème du flowshop à deux machines avec des opérations couplées en présence des temps de latence exacts, pour minimiser la date de fin de traitement des tâches. Comme le problème est NP-difficile, nous avons proposé des méthodes approchées, qui se résument dans les heuristiques et les métaheuristiques. Ainsi, nous avons proposé plusieurs heuristiques qui utilisent les règles de priorité LPT et SPT suivant les valeurs de a_j, L_j, b_j, et c_j, et la règle de Johnson. Pour évaluer la performance des heuristiques, nous avons réalisé des expérimentations numériques, puis nous avons comparé les résultats obtenus. D'après ces tests, nous avons constaté que les heuristiques à base de la règle LPT, et utilisant l'algorithme en parallèle pour le calcul du makespan fournissent de meilleurs résultats par rapport aux autres heuristiques. Toujours pour le même problème, nous avons proposé deux métaheuristiques, le recuit simulé (RS) et l'optimisation par essaims particulaires (OEP). Dans l'algorithme de l'OEP, nous avons introduit les opérateurs de mutation et de croisement. En outre, nous avons élaboré une méthode hybride en combinant ces deux dernières approches et nous avons comparé les résultats. D'après les expérimentations, nous avons noté que la méthode hybride est plus performante par rapport aux deux autres métaheuristiques.

CONCLUSION GÉNÉRALE

Dans cette thèse, nous nous sommes intéressés à l'étude du problème du flowshop dans l'objectif de minimiser la date de fin de traitement des tâches. Notre problème consiste à ordonnancer un ensemble de tâches dans un atelier de type flowshop à deux machines. Ces tâches sont composées d'un couple d'opérations sur la première machine et d'une seule opération sur la deuxième machine. Chaque deux opérations du couple d'opérations sont séparées par un temps de latence ou un délai exact.

Dans un premier temps, nous avons abordé notre travail par des définitions et des concepts de base en ordonnancement.

Dans le deuxième chapitre, nous avons mené une étude bibliographique portant sur les travaux existants concernant le problème d'ordonnancement avec la contrainte des temps de latence. Nous avons présenté les principaux problèmes d'ordonnancement étudiés dans la littérature dans le cas du flowshop en présence de la contrainte des temps de latence en distinguant, les problèmes avec des temps de latence minimaux, les problèmes avec des temps de latence maximaux, les problèmes avec des temps de latence minimaux et maximaux, et les problèmes avec des temps de latence exacts.

Dans le troisième chapitre, nous nous sommes focalisés sur la description du problème étudié. Nous avons aussi étudié la complexité de ses sous problèmes. Nous

avons prouvé que certains sont NP-difficiles et d'autres sont résolubles en des temps polynomiaux suite à la variation des valeurs des durées opératoires et des temps de latence. Nous avons terminé ce chapitre par une synthèse sur la complexité des problèmes déterminés à partir du problème symétrique à notre problème. Ce dernier consiste en un problème du flowshop à deux machines, telle que chaque tâche est composée d'une seule opération sur la première machine et d'un couple d'opérations sur la deuxième machine. Les opérations de chaque couple sont séparées par un temps de latence exact.

Le quatrième chapitre est dédié à la résolution du problème général prouvé NP-difficile dans le troisième chapitre. Pour commencer, nous avons proposé des bornes inférieures pour ce problème. Ensuite, nous avons proposé des méthodes qui nous permettent sa résolution approchée. Ainsi, nous avons présenté des approches heuristiques à la base des règles de priorités (LPT et SPT) et de l'algorithme de Johnson. Nous avons testé ces heuristiques sur des instances générées aléatoirement et comparé les résultats entre eux et avec la borne inférieure proposée. D'autres part, nous avons proposé des métaheuristiques, le recuit simulé et l'optimisation par essaims particulaires (OEP), dans laquelle nous avons introduit les opérateurs de mutation et de croisement. Pour améliorer la convergence de l'OEP, nous avons combiné l'OEP avec le recuit simulé. Pour tester l'efficacité de ces approches, nous avons réalisé des expérimentations numériques et nous avons comparé les résultats obtenus. Les résultats expérimentaux ont confirmé la bonne performance de la méthode hybride.

En perspective, il serait intéressant d'étudier la complexité du problème en intégrant d'autres contraintes oubien d'étudier le problème en considérant d'autres critères réguliers. Il serait également important d'étudier le problème en considérant des opérations couplées sur les deux machines. Un autre axe de recherche pour poursuivre ce travail est de proposer une méthode exacte de type Branch and Bound pour trouver une solution exacte.

BIBLIOGRAPHIE

[1] Adiri I. and Pohoryles D., *Flow-shop/no-idle or no-wait scheduling to minimize the sum of completion time*. Naval Research Quarterly, 29 : 495-504, 1982.

[2] Ageev A.A. and Baburin A.E., *Approximation algorithms for UET scheduling problems with exact delays*. Operations Research Letters, 35 : 533-540, 2007.

[3] Ageev A.A. and Kononov A.V., *Approximation algorithms for scheduling problems with exact delays*. In WAOA, 1-14, 2006.

[4] Aggoune R. and Portmann M.C., *Flow shop scheduling with limited machine availability : heuristic approach*. International Journal of Production Economics, 99 : 4-15, 2006.

[5] Aggoune R., *Minimizing the makespan for the flow shop scheduling problem with availability constraints*. European Journal of Operational Research, 153 : 534-543, 2004.

[6] Ahr D., Békési J., Galambos G., Oswald M. and Reinelt G., *An exact algorithm for scheduling identical coupled tasks*. Mathematical Methods of Operational Research, 59(11) : 193-203, 2004.

[7] Amrouche K. and Boudhar M., *Two machines flow shop with reentrance and exact time lag*. RAIRO Operations Research, 50 : 223-232, 2016.

[8] Ashour S.A., *A branch-and-bound algorithm for the flow shop scheduling problem*. AIIE Transactions, 2 : 172-176, 1970.

[9] Balas E., Lenstra J.K. and Vazacopoulos A., *The one machine with delayed precedence constraints and its use in job shop scheduling*. Management Science, 41(1) : 94-109, 1995.

[10] Baptiste P., *A note on scheduling identical coupled tasks in logarithmic time*. Discrete Applied Mathematics, 158 : 583-587, 2010.

[11] Baptiste P. and Hguny L.K., *A branch and bound algorithm for the* $F/no-idle/C_{max}$. In Proceedings of the International Conference On industrial Engineering and Production Management, IEPM'97, Lyon, 1 : 429-438, 1997.

[12] Békési J., Galambos G., Oswald M. and Reinelt G., *Improved analysis of an algorithm for the coupled task problem with UET jobs*. Operations Research Letters, 37 : 93-96, 2009.

[13] Bellman R., Esogbue A.O. and Nabeshima I., *Mathematical aspects of scheduling and applications*. Pergamon press, Oxford, 1982.

[14] Bellman R., *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[15] Billaut J.C. and Lopez P., *Enumeration of all optimal sequences in the two-machine fowshop*. Computational Engineering in Systems Application (CESA'98), Symposium on Industrial and Manufacturing Systems, IEEE-SMC/IMACS, 378-382, Nabeul-Hammamet, Tunisie, 1998.

[16] Blazewicz J., Ecker K., Kis T., Potts C.N., Tanas M. and Whitehead J., *Scheduling of coupled tasks with unit processing times*. Journal of Scheduling, 13 : 453-461, 2010.

[17] Blazewicz J., Breit J., Formanowicz P., Kubiak W. and Schmidt G., *Heuristics algorithms for the two-machine flowshop with limited machine availability*. Omega, 29, 599-608, 2001.

[18] Bouquard J.L. and Lente C., *Two-machine flow shop scheduling problems with mimimal and maximal delays*, 40R, 4 : 15-28, 2006.

[19] Brauner N., Finke G., Lehoux-Lebacque V., Potts C. and Whitehead J., *Scheduling of coupled tasks and one-machine no-wait robotic cells*, Computers and Operational Research, 36 (2) : 301-307, 2009.

[20] Breit J., *An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint*. Information Processing Letters, 90 : 273-278, 2004.

[21] Brucker P. and Knust S., *Complex Scheduling*. Second Edition, Springer, 2012.

[22] Brucker P., Knust S. and Oguz C., *Scheduling chains with identical jobs and constant delays on a single machine*. Mathematical Methods of Operations Research, 63 : 63-75, 2006.

[23] Brucker P., Knust S., Cheng T.C.E. and Shakhlevich N.V., *Complexity results for flow shop and open shop scheduling problems with transportation delays*. Annals of Operations Research, 129 : 81-106, 2004.

[24] Brucker P., Hilbig T. and Hurink J., *A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags*. Discrete Applied Mathematics, 94 : 77-99, 1999.

[25] Brucker P. and Knust S., *Complexity results for single-machine problems with positive finish-start time-lags*. Computing, 63 : 299-316, 1999.

[26] Campbell H.G., Dudek R.A. and Smith M.L., *A new heuristics for the n jobs m machines sequencing problem*. Management Science, 16 : 630-637, 1970.

[27] Carlier J. and Rebai I., *Two branch and bound algorithms for the permutation flow shop problem*. EJOR, 90 : 238-251, 1996.

[28] Carlier J. and Pinson E., *An algorithm for solving the job shop problem*. Management Science, 35 (2) : 164-176, 1989.

[29] Cheng T.C.E. and Liu Z., *3/2-approximation for two-machine no-wait flowshop scheduling with availability constraints*. Information Processing Letters, 88 : 161-165, 2003.

[30] Cheng T.C.E. and Liu Z., *Approximability of two-machine no-wait flowshop scheduling with availability constraints*. Operations Research Letters, 31 : 319-322, 2003.

[31] Cheng T.C.E. and Wang G., *An improved heuristic for two-machine flowshop scheduling with an availability constraint*. Operations Research Letters, 26 : 223-229, 2000.

[32] Cheng J., Kise H. and Matsumoto H., *A branch-and-bound algorithm with fuzzy inference for a permutation flowshop scheduling problem*. EJOR, 96 : 578-590, 1997.

[33] Chevalier A., *la programmation dynamique*. Dunod, 1977.

[34] Chu C. and Proth J.M., *Single machine scheduling with chain structured precedence constraints and separation time windows*. IEEE transactions on robotics and automation, 12 (6) : 835-844, 1996.

[35] Companys R., *Note on an improved branch-and-bound algorithm to solve $n/m/P/F_{max}$ problems*. TOP, 7 (1) : 25-31, 1999.

[36] Condotta A. and Shakhlevich N.V., *Scheduling coupled operation jobs with exact time lags*. Discrete Applied Mathematics, 160 : 2370-2388, 2012.

[37] Cook S.A., *The complexity of theorem-proving procedures*. In Third ACM Symposium on Theory of Computing, 151-158, 1971.

[38] Dannenbring D.G., *An evaluation of flow shop sequencing heuristics*. Management science, 2 : 1174-1182, 1977.

[39] Dell'Amico M., *Shop problems with two machines and time lags*. Operations Research, 44 : 777-787, 1996.

[40] Dell'Amico M. and Vaessens R.J.M., *Flow and open shop scheduling on two machines with transportation times and machine-independent processing times is NP-hard*. Materiali di discussione 141, Dipartimento di Economia Politica, Universita di Modena, 1996.

[41] Deppner F., *Ordonnancement d'atelier avec contraintes temporelles entre opérations*. Thèse de Doctorat, Institut National Polytechnique de Lorraine, 2004.

[42] Deppner F., *Ordonnancement d'ateliers avec contraintes d'écart minimal et maximal entre opérations*. Quatrième Conférence Francophone de MOdélisation et SIMulation, 430-436, 2003.

[43] Dréo J., Pétrowski A., Siarry P. et Taillard E., *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[44] Ecker K. and Tana M., *Complexity of scheduling of coupled tasks with chains precedence constraints and any constant length of gap*. Journal of the Operational Research Society, 63 (4) : 524-529, 2012.

[45] Espinouse M.L., Formanowics P. and Penz B., *Minimizing the makespan in the two-machine no-wait flow-shop with limited Availability*. Computer and Industrial Engineering, 37 : 497-500, 1999.

[46] Finta L. and Liu Z., *Single machine scheduling subject to precedence delays*. Discrete Applied Mathematics, 70 : 247-266, 1996.

[47] Fondrevelle J., Oulamara A. and Portmann M.C., *Permutation flow shop scheduling problems with time lags to minimize the weigted sum of machine completion times*. International Journal of Production Economics, 112 : 168-176, 2008.

[48] Fondrevelle J., Oulamara A. and Portmann M.C., *Permutation flowshop scheduling problem with maximal and minimal time lags*. Computers and Operations Research, 33 : 1540-1556, 2006.

[49] Fondrevelle J., *Résolution exacte de problèmes d'ordonnancement de type flow shop de permutation en présence de contraintes d'écarts temporels entre opérations*. Thèse de Doctorat, Institut National Polytechnique de Lorraine, France, 2005.

[50] Garey M. and Johnson D., *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[51] Garey M.R., Johnson D.S. and Sethi R., *The complexity of flow shop and job shop scheduling*. Mathematics of Operations Research, 1 : 117-129, 1976.

[52] Gilmore P.C. and Gomory R.E., *Sequencing a one state variable machine : a solvable case of the traveling salesman problem*. Operations Research, 12 (5) : 655-679, 1964.

[53] Goldberg D., *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, USA, 1989.

[54] Graham R.L., Lawler E.L., Lenstra J.K. and Rinnooy Kan A.H.G., *Optimization and approximation in deterministic sequencing and scheduling : a survey*. Annals of Discrete Mathematics, 5 : 287-326, 1979.

[55] Gupta J.N.D., *Comprarative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time lags*. Journal of Global Optimization, 9 : 239-253, 1996.

[56] Huo Y., Li H. and Zhao H., *Minimizing total completion time in two-machine flow shops with exact delays*. Computers and Operations Research, 36 : 2018-2030, 2009.

[57] Hurink J. and Keuchel J., *Local search algorithms for a single-machine scheduling problem with positive and negative time-lags*. Discrete Applied Mathematics, 112 : 179-197, 2001.

[58] Hwang F.J. and Lin Bertrand M.T., *Coupled task scheduling on a single machine subject to a fixed job sequence*. Computer and Industrial Engineering, 60 : 690-698, 2011.

[59] Ignall E. and Schrage L.E., *Application of the branch-and-bound algorithm to some flow shop problems*. Operations Research, 13 : 400-412, 1965.

[60] Janczewski R. and Kubale M., *Scheduling unit execution time tasks with symmetric time lags*. Journal of Applied Computer Science, 9 (2) : 45-51, 2001.

[61] Johnson S.M., *Optimal two and three stage production schedules with setup time included*. Naval Research Logistics Quarterly, 1 : 61-68, 1954.

[62] Joseph Y-T.L., Li H. and Zhao H., *Scheduling two machine flow shops with exact delays*. International Journal of Foundations of Computer Science, 18(2) : 341-359, 2007.

[63] Kamburowski J., *More on three-machine no-idle flow shops*. Computers and Industriel Engineering, 46 : 461-466, 2004.

[64] Karsan A.H. and Karimi B., *A discrete particle swarm optimization algorithm for scheduling parallel machines*. Computer and Industrial Engineering, 56 : 216-223, 2009.

[65] Kennedy J. and Eberhart R.C., *Particle swarm optimization*. In : Proceedings of IEEE international conference on neural networks, Piscataway, 1942-1948, 1995.

[66] Kern W. and Nawijn W.M., *Scheduling multi-operation jobs with time lags on a single machine*. Proceedings 2nd Twente Workshop on Graphs and Combinatorial Optimization, In Faigle U. and Hoede C. (eds.), Enschede, 1991.

[67] Kim C.W., Tanchoco J.M.A. and Koo P.H., *Deadlock prevention in manufacturing system with AGV system : Banker's algorithm approach*. Journal of Manufacturing Science and Engineering, 119 : 849-854.

[68] Kirkpatrick S., Gelatt C. and Vecchi M., *Optimzation by simulated annealing*. Science, 4598 : 671-680, 1983.

[69] Khurana K. and Bagga P.C., *Minimizing the makespan in a 2-machine flow shop with time lags ans setup condition*. Zeitschrift fur Operation Research, 28 : 163-174, 1984.

[70] Kubiak W., Balzewicz J., Formanowicz P., Breit J. and Schmidt G., *Two-machine flow shops with limited machine availability*. European Journal of Operational Research, 136 : 528-540, 2002.

[71] Lee C.Y., *Two-machine flow shop scheduling with availability constraints*. European Journal of Operational Research, 114 : 420-429, 1999.

[72] Lee C.Y., *Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint*. Operations Research Letters, 20 : 129-139, 1997.

[73] Lenstra J.K., Private communication, 1991.

[74] Li H. and Zhao H., *Scheduling coupled tasks on a single machine*. IEEE Symposium on Computational Intelligence in Scheduling, 1 (5), 137-142, 2007.

[75] Lin C.K.Y. and Haley K.B., *Scheduling two phase jobs with arbitrary time lags in a single machine system*. IMA Journal of Mathematic, 5 : 143-161, 1994.

[76] Lin S., *Computer Solution of the travelling Salesman problem*. Bell System Technical Journal, 44 : 2245-2269, 1965.

[77] Lomnicki Z.A., *A branch and bound algorithm for the exact solution of three-machine scheduling problems*. Operational Research Quarterly, 16 : 89-100, 1996.

[78] Lopez P. et Roubellat F., *Ordonnancement de la production*. Hermes science publication, 2001.

[79] Lourenço H.R., *A polynomial algorithm for special case of the one machine scheduling problem with time-lags*. Economics Working Papers Series, Departement of Economics and Business, Universitat Pompeu Fabra, 399, 1998.

[80] MacCarty B.L. and Liu J., *Adressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling*. International Journal of Production Reaserch, 31 (1) : 59-79, 1993.

[81] Meziani N., Boudhar M. and Oulamara A., *PSO and SA for the two machines flowshop scheduling problem*. European Journal of Industrial Engineering. Accepted 2017.

[82] Meziani N., Oulamara A. and Boudhar M., *Scheduling the two-machine flowshop problem with coupled-operations*. Annals of Operations Research. Second revision, submitted 2016.

[83] Meziani N., Boudhar M. et Oulamara A., *OEP et recuit simulé pour le problème du flowshop à deux machines avec des opérations couplées*. 17éme congrès annuel de la Société Française de Recherche Opérationnelle et Aide à la Décision, ROADEF'2016, Compiègne (France), 10-12 Février, 2016.

[84] Meziani N., Oulamara A. and Boudhar M., *NP-complétude du problème du flowshop à deux machines avec des opérations couplées*. X^{th} International Conference on Integrated Design and Production, CPI'2015, Tangier (Morocco), December 2-4, 2015.

[85] Meziani N., Boudhar M. and Oulamara A., *PSO and Simulated annealing for the two machines flowshop scheduling problem with coupled-operations*. Conference-School on Discrete Mathematics and Computer Science, DIMACOS'2015, Sidi Bel Abbès (Algérie), 15-19 Novembre, 2015.

[86] Meziani N., Oulamara A. and Boudhar M., *Problème du flowshop à deux machines avec des opérations couplées sur la première machine*. 6^{th} Operational Research Practice in Africa Conference, ORPA'2015, USTHB, Algiers (Algeria), April 20-22, 2015.

[87] Meziani N., Oulamara A. and Boudhar M., *Problème du flowshop à deux machines avec des opérations couplées sur la première machine*. 16éme édition du Congrès Annuel de la Société Française de la Recherche Opérationnelle et d'aide à la Décision, ROADEF'2015, Marseille (France), 25-27 Février, 2015.

[88] Meziani N., Oulamara A. and Boudhar M., *NP-hardness of the two-machine flowshop problem with coupled-operations*. Les annales RECITS 1, p. 11-26, 2014. ISSN : 2392-5426. http ://www.lrecits.usthb.dz/annales.htm.

[89] Meziani N., Oulamara A. and Boudhar M., *PSO for the two machines flowshop with the coupled-tasks*. Dixième édition du colloque sur l'Optimisation et les Systèmes d'Information (COSI'2013), Alger (Algérie), 9-11 Juin, 2013.

[90] Meziani N., Oulamara A. and Boudhar M., *Resolution of the coupled-tasks problem on two machines flowshop with the particle swarm optimization.* Premier Symposium International en Recherche Opérationnelle et ses Applications (ISORAP'2013), Marrakech (Maroc), 08-10 Mai, 2013.

[91] Meziani N., Oulamara A. and Boudhar M., *Minimizing the makespan on two-machines flowshop scheduling problem with coupled-tasks.* Neuvième Colloque sur l'Optimisation et les Systèmes d'Information, COSI'2012, Tlemcen (Algérie), 12-15 Mai, 2012.

[92] Mitten L.G., *Sequencing n Jobs on Two Jobs with Arbitrary Time Lags.* Management Science, 5 (3) : 293-298, 1958.

[93] Moukrim A., Rebaine D. and Serairi M., *A branch an bound algorithm for the two-machine flowshop problem with unit-time operations and time delays.* RAIRO Operations Research, 48 : 235-254, 2014.

[94] Munier A. and Rebaine D., *Polynomial time algorithms for the UET permutation flowshop problem with time delays.* Computers and Operations Research, 35 (2) : 525-537, 2008.

[95] Munier A. and Sourd F., *Scheduling chains on a single machine with non-negative time lags.* Mathematical Methods of Operations Research, 57 : 111-123, 2003.

[96] Orman A.J. and Potts C.N., *On the complexity of coupled-Task Scheduling.* Discrete Applied Mathematics, 72 : 141-154, 1997.

[97] Pawel J.K. and Kamburowski J., *On no-wait and no-idle flow shops with makespan criterion.* European Journal of Operational Research, 178 : 677-685, 2007.

[98] Potts C.N. and Whitehead J.D., *Heuristics for a coupled operation scheduling problem.* Journal of the Operational Research Society, 58 (10) : 1375-1388, 2007.

[99] Potts C.N., *An adaptive branching rule for the permutation flow-shop problem.* European Journal of Operational Research, 5 : 19-25, 1980.

[100] Quan-Ke P., Tasgetiren M.F. and Liang Y.C., *A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem.* Computer and Operational Research, 35 : 2807-2839, 2008.

[101] Ramon C. and Manelo M., *Different bahaviour of a double branch and bound algorithm on $Fm/prmu/C_{max}$ and $Fm/block/C_{max}$ problems.* Computers and Operations Research, 34 : 938-953, 2007.

[102] Rayward-Smith V.J. and Rebaine D., *Analysis of heuristics for the UET two-machine flow shop problem with time delays.* Computers and Operations Research, 35 : 3298-3310, 2008.

[103] Rayward-Smith V.J. and Rebaine D., *UET flow shop sheduling with delays.* Theoretical Informatics and Applications, 30(1) : 23-30, 1997.

[104] Rebaine D., *Flow shop vs. permutation shop with time delays.* Computers and Industrial Engineering, 48 : 357-362, 2005.

[105] Riezebos J. and Gaalman G.J.C., *Time lag size in multiple operations flow shop scheduling heuristics.* European Journal of Operational Research, 105 : 72-90, 1998.

[106] Riezebos J., Gaalman G.J.C. and Gupta J.N., *Flow shop scheduling with multiple operations and time lags.* Journal of Intelligent Manufacturing, 6 : 105-115, 1995.

[107] Saadani H., Guinet A. and Moalla M., *A travelling salesman approach to solve the $F/no-idle/C_{max}$ problem.* European Journal of Operational Research, 161 : 1-20, 2005.

[108] Saadani H., Guinet A. and Moalla M., *Three stage no-idle flow-shops.* Computers and Industrial Engineering, 44 : 425-434, 2003.

[109] Sha D.Y. and Cheng Y.H., *Hybrid particle swarm optimization for job shop scheudling problem.* Computer and Industrial Engineering, 51 : 791-808, 2006.

[110] Shapiro R.D., *Scheduling Coupled Tasks.* Naval Research logistics quartely, 20 : 489-498, 1980.

[111] Sheen G.J. and Liao L.W., *A branch and bound algorithm for the one-machine scheduling problem with minimum and maximum time lags.* European Journal of Operational Research, 181 : 102-116, 2007.

[112] Simonin G., Giroudeau R. and König J.C., *Complexity and approximation for scheduling problem for a torpedo.* Computer and Industrial Engeneering, 61 : 352-356, 2011.

[113] Simonin G., Giroudeau R. and König J.C., *Polynomial-time algorithms for scheduling problem for coupled-tasks in presence of treatment tasks*. Electronic Notes in Discrete Mathematics, 36 : 647-654, 2010.

[114] Simonin G., *Impact de la contrainte d'incompatibilité sur la complexité et l'approximation des problèmes d'ordonnancement en présence de tâches-couplées*. Thèse de Docorat, Laboratoire d'Informatique de Robotique et de Micróelectronique de Montpellier, France, 2009.

[115] Taillard E., *Some Efficient Heuristic Methods for the flow shop Sequencing Problem*. European Journal of Operational Research, 47 : 65-74, 1990.

[116] Tasgetiren M.F., Liang Y.C., Sevkli M., Gencyilmaz G., *A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem*. European Journal of Operational Research, 177 : 1930-1947, 2007.

[117] Tasgetiren M.F., Sevkli M., Liang Y.C. and Gencyilmaz G., *particle swarm optimisation algorithm for single machine total weighted tardiness problem*. In : proceeding of the IEEE confgress on evoultionary computation, 2 : 1412-1419, Portland, 2004.

[118] Tasgetiren M.F., Sevkli M., Liang Y.C. and Gencyilmaz G., *Particle swarm optimization algorithm for makespan and maximum lateness minimization in permutation flowshop sequencing problem*. In : 4th international symposium on intelligent manufacturing systems, 431-441, Turkey, 2004.

[119] Wang G. and Cheng T.C.E., *Heuristics for two-machine no-wait flowshop scheduling with an availability constraint*. Information Processing Letters, 80, 305-309, 2001.

[120] Widmen M. and Hetz A., *A new Heuristic Method for the Flow Shop Sequencing Problem*. European Journal of Operational Research, 41, 186-193, 1989.

[121] Wikum E.D., Llewellyn D.C. and Nemhauser G.L., *One machine generalized precedence constrained scheduling problems*. Operations Research Letters, 16 : 87-99, 1994.

[122] Wlodzimierz S., *The flow shop problem with time lags and separated setup times*. Zeitschrift Operations Research, 30 : 15-22, 1986.

[123] Xiandong Z. and Steef van de V., *Polynomial-time approximation schemes for scheduling problems with time lags*. Journal of Scheduling, 13 : 553-559, 2010.

[124] Xianpeng W. and Lixin T., *A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking*. Applied Soft Computing, 12 : 652-662, 2012.

[125] Yang D.L. and Chern M.S., *A two-machine flowshop sequencing problem with limited waiting time constraints*. Computers and Industrial Engineering, 28 : 63-70, 1995.

[126] Yu W., Hoogeveen H. and Lenstra J.K., *Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard*. Journal of Scheduling, 7 : 333-348, 2004.

[127] Yu W., *The two-machine flow shop problem with delays and the one-machine total tardiness problem*. PHD Thesis, Technische Universiteit Eindhoven, 1996.

[128] Zhigang L., Xingsheng G. and Bin J., *A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan*. Applied Mathematics and Computation, 175 : 773-785, 2006.

[129] Zhigang L., Bin J. and Xingsheng G., *A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan*. Applied Mathematics and Computation, 183 : 1008-1017, 2006.