N° d'ordre :06/2021 ;-D/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumedienne

Faculté de mathématiques

Thèse de Doctorat en Sciences présentée pour l'obtention du diplôme de docteur
Spécialité : Mathématiques
Option : Recherche Opérationnelle
Par :

TIGANE Meriem

Thème

Ordonnancement et Gestion de l'Énergie

Soutenue publiquement le : 06 Février 2021, devant le jury :

M^r.	Belbachir	Hacène	Professeur, U.ST.H.B	Président
M^r.	BOUDHAR	Mourad	Professeur, USTHB	Directeur de Thèse
M^r.	DAHANE	Mohammed	Maitre de Conférence/A, U. Lorraine (France)	Co-directeur de Thèse
M^r.	Aitzai	Abdelhakim	Professeur, USTHB	Examinateur
M^r.	Berrichi	Ali	Professeur, UMB Boumerdes	Examinateur
M^r.	Boulif	Menouar	Professeur, UMB Boumerdes	Examinateur

Résumé

Au cours des dernières décennies, les préoccupations croissantes sur le changement climatique et la diminution des ressources en énergie fossile ont mis en avant l'importance de la question sur la consommation énergétique.

Le sujet de cette thèse porte sur les problèmes d'ordonnancement bi-objectifs qui consistent à planifier l'exécution de tâches sur des machines avec comme fonction objectif secondaire la minimisation de la consommation totale d'énergie.

Dans ces problèmes, on considère que le temps d'exécution des tâches augmente avec le temps. Un tel modèle peut être rencontré en pratique par exemple : la production des laminoirs d'acier ou dans la planification des opérations médicales en considérant l'état de santé des patients.

On étudie le problème d'ordonnancement de tâches déteriorables sur des machines parallèles générales. La déterioration des machines est aussi considérée. On présente un modèle mathématique et un algorithme énumératif. Une approche basée sur l'algorithme NSGA-II et des variantes sont proposées. La technique TOPSIS a été utilisée pour ordonner les front pareto approché. Enfin, l'efficacité des algorithmes proposés est étudiée.

On étudie aussi le problème d'ordonnancement de tâches déteriorables dans un système flow shop hybride. On présente un modèle mathématique. On propose des algorithmes basés sur le NSGA-II et le recuit simulé et on compare leur efficacité.

Mots clés : Ordonnancement, énergie, tâches déteriorables, NSGA-II, TOPSIS

Abstract

Over the years, managing energy consumption has become crucial with the growing interest in climate change and the diminishing in non-renewable energy resources.

This thesis deals with bi-objective scheduling problems consisting on planning the process of tasks over a set of machines with energy consumption as a secondary objective.

In these models, we considered that the processing times are varying over time. Example of such models can be seen in the scheduling of steel rolling mills or medical procedure under health conditions.

We considered first, a scheduling problem with deteriorating jobs on unrelated parallel machine where the machines are subject to deterioration effect. We present an enumerative algorithm and several NSGA-II based algorithms. We use the TOPSIS method to order the near-pareto solution. We compare the efficiency of the metaheuristics.

We also studied a scheduling problem with deteriorating jobs on a flexible flow shop. We present a mathematical formulation. We present an NSGA-II based algorithm and a simulated annealing based algorithm. The efficiency of the two algorithms are compared.

Keywords : Scheduling, Energy, unrelated machines, flexible flowshop, Deteriorating jobs, NSGA-II, Topsis

Remerciements

La réalisation de cette thèse a été faite grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude. Je voudrais dans un premier temps remercier mon directeur de thèse Mr Boudhar, professeur à l'université USTHB, pour avoir accepté de m'encadrer et d'avoir proposé la thématique. Je le remercie pour sa supervision, ses conseils avisés et ses encouragements tout au long de la rédaction de cette thèse.

Je remercie également mon co-directeur de thèse Mr Dahane, maitre de conférence HDR à l'ENIM, pour m'avoir encadré, orienté mes recherches et aidé à la rédaction de mon premier article, ainsi que pour sa patience.

J'adresse mes remerciements à Mrs Aitzai, professeur à l'USTHB, Berrichi et Boulif, professeurs à l'UMB Boumerdes, de l'honneur qu'ils m'ont fait en acceptant d'être examinateurs de cette thèse et à Belbachir, professeur à l'USTHB, de présider le jury de soutenance.

J'adresse mes sincères remerciements à tous les chercheurs et toutes les personnes qui ont pris le temps de discuter de mon sujet et ont accepté de répondre à mes questions durant mes recherches.

Enfin, je remercie mes chers parents et mon frère pour leurs encouragements et soutien inconditionnel. Je remercie mes amis pour leur soutien constant.

Table des matières

Table des figures

Liste des tableaux

Introduction

> Il y a quarante ans, on nous la promettait pour dans quarante ans ; aujourd'hui, on nous la promet encore pour dans quarante ans. Ça doit être une des constantes fondamentales de la physique.
> *21 Energies renouvelables insolites pour le 21^e siècle*
>
> – *Denis Bonnelle, Renaud de Richter*, 2010

La théorie de l'ordonnancement est une science mathématique dont l'application en industrie et en manufacture est la plus développée et est une branche de la recherche opérationnelle. Son rôle a été mis en avant au début du $XX^{\text{ème}}$ siècle et les premiers algorithmes d'ordonnancement sont attribués à Johnson, Jackson et Smith dans les années 50. Les problèmes d'ordonnancement consistent dans la plannification de l'exécution d'un ensemble de tâches et leur allocation à des ressources sous des contraintes temporelles et de disponibilité des ressources visant à optimiser certains critères. Le critère considéré peut être l'optimisation de la productivité, la minimisation du temps d'exécution total ou le nombre de tâches en retard. Les problèmes peuvent être mono-objectif, multi-objectif ou une combinaison linéaire de plusieurs objectifs.

La gestion de l'énergie est une question cruciale, car la production d'énergie est soumise à plusieurs contraintes : disponibilités des ressources, croissance de la demande et l'impact des énergies fossiles sur le réchauffement climatique. L'énergie a été et est au coeur des crises économiques, on peut citer la crise dite " de l'énergie" survenue en 1973 et la crise actuelle qui est précédée par une hausse des prix du pétrôle. Autant de raisons qui font de la gestion de l'énergie un problème primordial notamment dans le milieu industriel.

On retrouve dans la littérature, plusieurs problèmes d'ordonnancement qui traitent la gestion de l'énergie comme une contrainte ou un objectif à minimiser. Dans les modèles considérés, les chercheurs abordent les problèmes avec les définitions classiques des tâches, en particulier le temps d'exécution. Or dans la vie réelle, il existe des modèles où le temps d'exécution d'une tâches peut croître avec le temps, par exemple : la fabrication des laminoirs d'acier, l'exécution des opérations chirurgicales selon l'état de santé du patient. Dans d'autres cas, le temps d'exécution peut décroître : En agriculture où le temps de la moisson diminue avec la diminution de la récolte. On appelle de telles tâches : tâches déteriorables.

Dans cette thèse, on traîte des problèmes d'ordonnancement à deux objectifs, où le temps

d'exécution varie avec le temps en considérant que l'un des objectifs est un critère d'optimisation classique de l'ordonnancement et le second est la minimisation de la consommation énergétique. Cette thèse est divisée en deux parties :

Notions de bases Cette partie sert à familiariser le lecteur avec les notions abordées et est divisée en trois chapitres :

- **Optimisation multicritère :** On définit dans ce chapitre les problèmes d'optimisation multicritères en présentant les différentes méthodes de résolution.
- **Ordonnancement** Dans ce chapitre, on définit les notions relatives à la théorie de l'ordonnancement et on présente les principaux résultats concernant certains problèmes classiques d'ordonnancement.
- **Les tâches déteriorables** On présente la notion de tâche déteriorable ainsi qu'un état de l'art.

Ordonnancement et Gestion de l'Energie : Cette seconde partie est divisée en trois chapitres :

- **Gestion de l'Energie : Etat de l'art** Dans ce chapitre, on présente un état de l'art sur les problèmes d'ordonnancement ayant trait à la gestion de l'énergie.
- **Sur un problème d'ordonnancement sur machines parallèles générales** On considère dans ce chapitre, un problème d'ordonnancement sur machines parallèles générales avec tâches déteriorables avec comme objectif la minimisation simultannée du makespan et de la consommation d'énergie.

  On présente un modèle mathématique. On propose un algorithme énumératif ainsi que des approche métaheuristique basée sur le NSGA-II et quelques variantes de l'approche et on utilise la technique topsis pour trier les solutions obtenues. Enfin, on compare l'efficacité de ces approches.
- **Sur un problème d'ordonnancement dans un système flow shop** On étudie dans ce chapitre le problème de minimisation du makespan et de la consommation énergétique dans un système flow shop hybride avec des tâches déteriorables définies par une fonction non linéaire. On propose un modèle mathématique et des approches métaheuristiques en appliquant les approches MOMSA et NSGA-II et on compare leur efficacité.

On termine notre thèse par une conclusion qui résume l'ensemble des résultats obtenus et donne les perspectives à venir.

Première partie

Notions de base

Chapitre 1

Ordonnancement

Ce chapitre est une introduction à la théorie de l'ordonnancement présentant les notions de base de l'ordonnancement. Alharkan (2010), Baker (1974) ont été les ouvrages de référence pour ce chapitre.

1.1 Définition

Un problème d'ordonnancement peut être défini comme étant le "processus d'organisation, choix et programmation dans le temps de l'usage des ressources pour traiter toutes les activités nécessaires afin d'aboutir à un objectif donné tout en satisfaisant un ensemble de contraintes sur les activités et les ressources" (Morton and Pentico, 1993). Les activités sont appelées tâches.

Une ressource est un moyen technique ou humain nécessaire pour la réalisation des tâches. On distingue plusieurs types de ressources : les ressources renouvelables et les ressources consommables. Dans le cas des ressources renouvelables, on distingue deux types : Les ressources disjonctives, c-à-d : La ressource ne peut exécuter qu'une seule tâche à la fois et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément.

Un problème d'ordonnancement est généralement décrit comme suit : Soit donné un ensemble de n tâches $\{T_1, \ldots, T_n\}$ à traiter sur un ensemble de m machines, $(n, m) \in \mathbb{N}^{*2}$, on cherche l'ordonnancement qui satisfait un certain nombres de contraintes tout en optimisant un ou plusieurs objectifs.

La solution d'un problème d'ordonnancement est appelé ordonnancement. On dit qu'un ordonnancement est réalisable ou admissible si toutes les tâches ont sont traitées en respectant toutes les contraintes.

1.2 Tâches

Une tâche est définie par la donnée de certaines caractéristiques :

Date d'arrivée, de disponibilité (release time, ready time ou arrival time) : noté r_i et représente la date à laquelle la tâche T_i est prête à être traitée. Si toutes les tâches ont la même date d'arrivé, on pose $r_i = 0, \forall i \in \{1, \ldots, n\}$.

Taille ou temps d'exécution (work, size, execution time) : noté par p_i et représente le temps nécessaire à une machine pour traiter la tâche.

Date échue ou au plus tard (due date) : noté d_i et indique que le traitement de la tâche T_i doit être achevé avant la date d_i sous peine de pénalité.

Date limite ou absolue (deadline) : noté $\tilde{d}_i$ et indique que le traitement de la tâche T_i doit être impérativement achevé avant la date $\tilde{d}_i$.

temps d'installation (setup time) : Représente le temps nécessaire pour préparer une machine au traitement d'une tâche lorsqu'elle en termine une.

Poids (weight) : noté w_i et exprime la priorité relative de T_i.

Il existe deux types de tâches :

1. Les tâches récurrentes : appelées aussi périodiques. Les tâches récurrentes constituent un ensemble fini noté $T = \{T_{1_k}, T_{2_k}, \cdots, T_{n_k}\}$ où T_{i_k} est la tâche définie par : une date de disponibilité r_{ik} avec $r_{i1} \leq r_{i2} \leq \cdots \leq r_{ik} \cdots$, une taille $p_{i,k}$, une date échue $d_{i,k}$.
2. tâches non récurrentes : appelés aussi apériodiques.

1.3 Les machines

Le traitement des tâches se fait sur une seule ou plusieurs machines. Dans le cas où il y a plusieurs machines, on distingue deux types de machines :

Les machines parallèles : Ces machines exécutent le même traitement et sont donc capables de traiter n'importe quelle tâche. En fonction de la vitesse de traitement, on distingue 3 types de machines parallèles :

1. *Les machines identiques* : La même vitesse est utilisée pour toutes les machines.
2. *Les machines uniformes* : Chaque machine a sa propre vitesse qui est constante.
3. *Les machines générales* : La vitesse de la machine dépend de la tâche á traiter.

Les machines spécialisées : Ces machines ne peuvent exécuter que certaines tâches. Il existe 3 modes de traitement :

1. *Flow shop* : Les tâches doivent être traitées par toutes les machines et dans le même ordre.
2. *Open shop* : Les tâches doivent être traités par toutes les machines mais sans aucun ordre.
3. *Job shop* : Le sous-ensemble devant traiter une tâche et l'ordre de traitement des tâches est arbitraires. mais doivent être spécifiés à l'avance.

1.4 Les contraintes

On énumère dans ce qui suit certaines contraintes qu'on rencontre dans les problèmes d'ordonnancement classiques :

Disponibilité : Chaque tâche T_i est traitée dans l'intervalle $[r_i, +\infty[$ lorsque aucune date limite n'est définie, sinon la tâche est traitée dans l'intervalle $[r_i, \tilde{d}_i[$.

Parallélisme : On parle de tâches parallèles lorsqu'il est possible de traiter chaque tâche par plusieurs machines simultanément. Dans les problèmes d'ordonnancement classiques, chaque tâche est exécuté par au plus une machine, à chaque instant $t \in [0, +\infty[$.

Préemption : Un ordonnancement peut être préemptif (ou avec préemption) si on peut suspendre le traitement d'une tâche et la reprendre plus tard, sans aucun coût. Dans le cas où la tâche peut être reprise sur une autre machine, on parle de migration. Sinon, on dit que le traitement est non préemptif ou sans préemption.

Précédence : Une contrainte de précédence peut être définie par une relation d'ordre. On écrit $T_i < T_j$ pour dire que la tâche T_i doit être achevée avant T_j. On dit que les tâches sont dépendantes si elles sont reliées par une relation d'ordre, sinon elles sont indépendantes. Dans le cas de machines spécialisées, les opérations constituant une tâche sont toujours dépendantes (sauf open shop) bien que les tâches peuvent être soit dépendantes soit indépendantes.

- Un ensemble de tâches ordonné par une relation de précédence est représenté par un graphe orienté dans lequel les sommets représentent les tâche et les arcs les contraintes de précédence.
- Une tâche T_i est dite disponible à la date t si $r_i \leq t$ et si à l'instant t, le traitement des tâches qui la précèdent est achevé.

1.5 Critère d'optimalité

Les paramètres suivants peuvent être calculés pour chaque tâche $T_i, i = \overline{1,n}$ traitée dans un ordonnancement donné :

Une date de fin d'exécution (completion time) C_i **:** date de fin de traitement de la tâche T_i.

Un temps de séjour (flow time) f_i **:** C'est la différence entre la date de fin d'exécution et la date d'arrivée $f_i = C_i - r_i$ et qui représente la somme du temps d'attente et du temps de traitement.

Une tardivité : décalage temporel ou retard algébrique (lateness) l_i : C'est la différence entre la date de fin d'exécution et la date échue $l_i = C_i - d_i$.

Le retard vrai (tardiness) : t_i calculée comme suit $t_i = \max_{1\leq i\leq n}\{l_i, 0\}$.

L'avance (Earliness) : E_i où $E_i = \max_{1\leq i\leq n}\{0, d_i - C_i\} = -\min_{1\leq i\leq n}\{0, l_i\}$.

Ces paramètres nous permettent de définir les critères suivants :

La durée totale de l'ordonnancement (Makespan) : représente la plus grande valeur prise par la date de fin d'exécution et est notée C_{max}, donnée par : $C_{max} = \max_{1\leq i\leq n}\{C_i\}$.

Le plus grand retard algébrique noté L_{max} et donné par la formule $L_{max} = \max_{1\leq i\leq n}\{L_i\}$.

Le plus grand temps de séjour : noté par F_{max} et donné par la formule $F_{max} = \max_{1\leq i\leq n}\{f_i\}$

Temps de séjour moyen : noté par $\bar{F}$ et donné par la formule $\bar{F} = \frac{1}{n}\sum_{1\leq i\leq n} f_i$

Le plus grand retard vrai : noté T_{max} et donné par la formule $T_{max} = \max_{1\leq i\leq n}\{t_i\}$

Le nombre de tâches en retard : noté N_T et donné par la formule $N_T = \sum_{1\leq i\leq n} u_i$.

le retard moyen noté $\bar{T}$ et est donné par la formule $\bar{T} = \frac{1}{n}\sum_{1\leq i\leq n} t_i$.

1.6 Classification des problèmes d'ordonnancement

Afin de simplifier la classification d'un problème d'ordonnancement, plusieurs notations ont été utilisées. La plus courante est la notation introduite par Graham et al. (1979) et reprise par Bazewicz et al. (2001). Toutes les données d'un problème d'ordonnancement sont résumées dans une notation à trois champs : $\alpha|\beta|\gamma$. On donne une description détaillée pour chaque champs dans ce qui suit :

Champs α : Les ressources . On y décrit les machines utilisées : leur type (le type d'atelier), leur nombre. Ce champs se divise en deux sous-champs et est désigné par $\alpha = \alpha_1\alpha_2$: α_1 : Décrit le type de machine.

$\alpha_1 = 1$ **:** Une seule machine.

$\alpha_1 = P$ **:** Machines parallèles identiques.

$\alpha_1 = Q$ **:** Machines parallèles uniformes.

$\alpha_1 = R$ **:** Machines parallèles générales.

$\alpha_1 = O$ **:** Machines spécialisées, système open shop.

$\alpha_1 = F$ **:** Machines spécialisées, système flow shop.

$\alpha_1 = J$ **:** Machines spécialisées, système job shop.

α_2 : Décrit le nombre de machines lorsque $\alpha_1 \neq 1$.

$\alpha_2 = \emptyset$ **:** Le nombre de machines est variable.

$\alpha_2 = m$ **:** Le nombre de machines est égal à m où $m > 1$.

Champs β : Les contraintes . On y décrit les tâches et les contraintes qui leur sont liées ainsi que les contraintes sur les ressources. Ce champs peut être divisé en plusieurs sous-champs. Les premiers concernent les tâches et les derniers sont réservés aux ressource. Quelques notations de contraintes utilisées.

- $pmtn$: Indique que les tâches sont préemptives.
- $prec$: Indique l'existence de contraintes de précédence et leur type.
- r_i : Indique les dates de disponibilité.
- p_i : Indique les tailles des travaux.
- les contrainte sur les ressources : contraintes de puissance, énergetiques...

Champs γ : Critères d'optimalité. Ce champs précise le critère d'optimalité utilisé.

L'objectif est de minimiser γ.

1.7 Représentation d'un ordonnancement : Diagramme de Gantt

Le diagramme de Gantt est une technique populaire de représentation graphique d'un ordonnancement de tâches sur machines. L'axe des abscisses représente le temps et les rectangles sur l'axe des ordonnées représentent les machines. Chaque barre horizontale montre les temps de début et de fin d'exécution d'une tâche. Le numéro de la tâche est inscrit dans le rectangle. La longueur du rectangle représente le temps d'exécution de la tâche. L'espace vide entre deux rectangles représentant les tâches indique le temps de repos de la machine.

1.8 Règles de passage

Une règle de passage ou de priorité sert à déterminer l'ordre dans laquelle les tâches sont traitées. On cite quelques règles de passage utilisées en ordonnancement :

- SPT ou SEPT (Shortest processing time, shortest expected processing time) : Les tâches sont traitées selon l'ordre croissant de leur temps d'exécution. Plusieurs variantes existent :
  1. SRPT (shortest remaining processing time),
  2. TSPT (Truncated SPT) : La tâche avec le plus petit temps d'exécution est choisie en premier mais s'il existe une tâche avec un temps d'attente supérieur à w, cette dernière est choisie en premier. w est une valeur choisie arbitrairement.
  3. WSPT (weighted SPT) : Les tâches sont traitées selon l'ordre croissant de $\frac{p_i}{w_i}$.
- LPT ou LEPT (Longest processing time, longest expected processing time) : Les tâches sont traitées selon l'ordre décroissant de leur temps d'exécution. Plusieurs variantes existent : TLPT (total LPT), LRPT (longest remaining PT),
- EDF (earliest deadline first) : Les tâches sont traitées selon l'ordre croissant de leurs dates limites,
- EDD (Earliest due date) : Les tâches sont traitées selon l'ordre croissant de leurs dates échues,
- FIFO, FCFS ou SORT (First in, first out ; first come, first served ou smallest release date) : La première tâche disponible est exécutée en premier,
- LIFO, LCFS (Last in, last out ou last come, first served) : Les tâches sont traitées selon l'ordre décroissant de leurs dates de disponibilité,
- CP (Critical path) : C'est une méthode d'ordonnancement prenant en compte des relations de dépendances entre tâches et qui permet de déterminer les dates au plus tôt et au plus tôt de début de traitement des tâches et de calculer la durée minimum de l'ordonnancement tout en respectant les contraintes temporelles,
- Round Robin : Attribue des tranches de temps à chaque travail en proportion égale, sans accorder de priorité aux tâches.

1.9 Catégories de problèmes d'ordonnancement

Les problèmes d'ordonnancement peuvent être divisés en plusieurs catégories selon le types de paramètres définis (tels que temps d'exécution, date de disponibilité, etc) :

- Déterministe : Lorsque tous les paramètres et données du problème sont connus par avance et sont fixes [Le nombre de tâches et leurs caractéristiques]. Parmi les problèmes déterministes on rencontre les problèmes **statiques** où chaque tâche est définie par une date de disponibilité.
- Dynamique : L'ensemble des tâches n'est pas fixée à l'avance et les tâches arrivent à des périodes différentes.

1.10 Combinatoire enumérative

Dans la plupart des problèmes d'ordonnancement, la résolution du problème consiste à l'enumération de l'ensemble des solutions possibles et d'en choisir la meilleure. Pour calculer la taille de l'ensemble des solutions possibles, on utilise les notions suivantes :

- Permutations : Une permutation de n éléments distincts est le réarrangement ordonné et sans répétition de ces éléments. Le nombre de permutations de n éléments distincts est égal à $n!$. Par exemple, le nombre de solutions possibles pour l'ordonnancement de n tâches sur une machine est égale à $n!$
- Arrangements : Un arrangement est une permutation de k éléments parmi n. Le nombre d'arragements de k éléments parmi n est noté $A_{n,k}$ et est donné par l'exmpression $A_{n,k} = \frac{n!}{(n-k)!}$.
- Combinaisons : Une combinaison de k éléments pris parmi n éléments distincts est un sous ensemble à k éléments distincts de cet ensemble. Les éléments sont pris sans répétition et ne sont pas ordonnés. Le nombre de combinaisons de k éléments parmi n est appelé coefficient binomial et est noté $\binom{n}{k}$ ou C_n^k et est donné par l'expression $C_n^k = \frac{n!}{(n-k)!k!}$.

Une généralisation du coefficient binomial est le coefficient multinomial. Soient $n \in \mathbb{N}$ et $s \in \{1, \ldots, n\}$. Soient $n_1, n_2, \ldots, n_s \in \mathbb{N}$ tels que $n_1 + n_2 + \ldots + n_s = n$, le coefficient multinomial est le nombre de partitions d'un ensemble de taille n en s sous-ensembles S_1, S_2, $\ldots$, S_s de tailles respectives n_1, n_2, $\ldots$, n_s et est noté $\binom{n}{n_1,n_2,\ldots,n_s}$. La formule explicite est $\binom{n}{n_1,n_2,\ldots,n_s} = \frac{n!}{n_1!n_2!\ldots n_s!}$. Pour $s = 2$, on retrouve le coefficient binomial.

1.11 Méthodes de résolution

Le choix de la méthode de résolution d'un problème d'ordonnancement dépend des besoins de l'ordonnanceur et des caractéristiques du problème. On peut diviser les méthodes en deux classes :

- Méthodes exactes : Ces méthodes permettent d'obtenir une solution optimale. Ce sont des méthodes d'énumération. Les méthodes les plus connues sont :

  La programmation dynamique ; qui consiste à résoudre des sous-problèmes du problème afin d'obtenir solution du problème global et est basée sur le principe d'optimalité de Bellman. Un exemple classique est la recherche du plus court chemin.

  La programmation linéaire ; introduite par George Dantzig et qui consiste à minimiser une fonction linéaire avec des contraintes linéaires.

  Séparation et évaluation ; (Branch and bound) Méthode d'énumération implicite qui consiste à décomposer le problème initial en sous problèmes pour former une partition de l'espace des solutions et évaluer les bornes optimales des solutions de chaque sous-problème.

  Ces méthodes de résolution sont parfois coûteuses en temps d'exécution qui est lié à la taille du problème et de sa complexité.
- Méthodes approchées : Ces méthodes ne fournissement pas forcément une solution optimale mais elles sont utilisées pour la recherche d'une solution satisfaisante en un laps de temps

raisonnable. Ces méthodes englobent

Heuristiques : elles sont spécifiques au problème donné. On peut citer l'algorithme glouton.

Métaheuristiques : elles peuvent être appliquées sur différents problèmes.

1.12 Problèmes d'ordonnancement : Etat de l'art

1.12.1 Machines parallèles

Les problèmes d'ordonnancement sur machines parallèles où la préemption est exclue sont en général NP-difficiles tels que $P||C_{max}$ et $R||C_{max}$. Pour le problème $P||C_{max}$, plusieurs heuristiques sont utilisées afin de déterminer une solution proche de l'optimale. Certaines heuristiques incluent la règle LPT décrite précédemment. On peut utiliser un algorithme de branch and bound pour le problème $R_m||C_{max}$.

Rabadi et al. (2006) ont proposé une heuristique Meta-Raps (meta-heuristic for randomized priority search) pour le problème NP difficile $R_m|setup\,time|C_{max}$ et ont formulé un modèle mathématique en nombres entiers mixtes. Le même problème a été étudié par Arnaout et al. (2014) où les auteurs ont proposé un algorithme colonie de fourmis à deux phases plus performant que Meta-RaPS.

Par contre, la difficulté s'estompe lorsqu'on accepte la préemption, par exemple, la procèdure McNaughton peut résoudre de façon optimale le problème $P|pmtn|C_{max}$ en un temps $O(n)$. Pour le problème avec contraintes de précédence sous forme d'arborescence, notée $P_m|p_i = 1, tree|C_{max}$, on peut utiliser la règle CP si l'arborescence est un arbre descendant ou ascendant.

Pour les machines uniformes, le problème $Q_m|pmtn|C_{max}$ peut être résolu en appliquant la règle LRPT et en assignant la tâche à la machine la plus rapide disponible. En changeant le critère d'optimalité en $\sum C_i$, on utilise la règle SRPT.

La table 1.1 retrace quelques recherches concernant la minimisation du makespan concernant les machines parallèles générales :

Problème	Référence
$R_m\|\|C_{max}$	Charalambous and Fleszar (2012), Fanjul-Peyro and Ruiz (2012), Garey and Johnson (1979), Hariri and Potts (1991), Kumar et al. (2011), Shchepin and Vakhania (2005)
$R_m\|st\|C_{max}$	Arnaout et al. (2014), Chang and Chen (2011), de Paula et al. (2007), Eroglu et al. (2014), Ezugwu et al. (2018)
$R_m\|ress\,constraint\|C_{max}$	Chen (2005)
$R_m\|r_i\|C_{max}$	Zhang et al. (2016)
$R_m\|d_i\|C_{max}$	Yang-Kuei and Chi-Wei (2013)
$R_m\|r_i, d_i, st\|C_{max}$; st=stup time	Nikabadi and Naderi (2016)
$R_m\|r_i, st, prec, ress\|C_{max}$	Afzalirad and Rezaeian (2016)

$R_m\|batch\|C_{max}$	Damodaran et al. (2012)
$R_m\|ma\|C_{max}$	Ebrahimi and Rezaeian (2015)

TABLE 1.1: Problèmes d'optimisation du makespan sur machines parallèles générales

1.13 Flow shop hybride

Le problème d'ordonnancement flow shop hybride est la combinaison de deux problèmes d'ordonnancement : Les problèmes d'ordonnancement flow shop et machines parallèles. On le définit comme suit : Il y a L niveaux. Chaque niveau l consiste en M^l machines parallèles ($M^l \in \mathbb{N}^*$). Au moins un étage doit contenir plusieurs machines. On a n tâches à exécuter. Chaque tâche consiste en L opérations où chaque opération l doit être exécutée par une machine au niveau l. Les tâches doivent être acheminées aux niveaux dans le même ordre de 1 vers L. La préemption n'est pas permise. Dans un niveau, les machines utilisées peuvent être identiques, uniformes ou générales. Ce problème est aussi appelé flow shop flexible.

Le problème flow shop hybride a été étudié en premier par Arthanary and Ramamurthy en 1971. Le problème a été prouvé NP-difficile au sens fort pour les critères d'optimisation classiques Bansal et al. (2011).

En utilisant la notation proposée par Billaut and Proust (1999) qui généralise celle de Graham et al. (1979), $\alpha|\beta|\gamma$ en divisant α en quatre termes i.e. $\alpha_1\alpha_2(\alpha_3\alpha_4^{(1)}, \alpha_3\alpha_4^{(2)}, \ldots, \alpha_3\alpha_4^{(\alpha_2)})$. On a : α_1 indique le problème considéré qui est noté par HF pour désigner le flow shop hybride. Le paramètre α_2 indique le nombre de niveaux qui est >1. Le paramètre α_3 donne le type de machines sur chaque niveau, c-à-d. identiques (P), uniformes (Q), générales (R) ou 0 s'il n'y a qu'une seule machine et α_4 indique leur nombre. S'il y a plusieurs niveaux avec le même type et nombre de machines, les termes peuvent être groupés comme suit : $((\alpha_3\alpha_4)^l)_{l=s}^k$ où s et k sont les indices du premier et dernier niveaux concernés, respectivement.

Par exemple, la notation $HF2(P2^1, Q3^2)$ indique qu'il s'agit d'un système flow shop hybride à deux niveaux. Le premier niveau contient deux machines parallèles identiques et le second niveau contient 3 machines uniformes.

On commence par donner la table 1.2 présentant quelques travaux concernant les problèmes d'ordonnancement dans des ateliers flowshop classiques.

Problème	Référence
C_{max}	Baskar and Xavior (2012), Chakraborty and Laha (2007), Modrak and Pandian (2010), Nowicki and Smutnicki (1996), Stutzle (1997), Gary et al.[1976], Ignall and Schrage (1965)
No-wait, C_{max}	Chaudhry et al. (2014), Nailwal et al. (2016), Zandieh and Rashidi (2009)
$\sum C_{ij}$	Tseng and Lin (2009)

no-wait, $\sum C_{ij}$	Sapkal and Laha (2013), Ye et al. (2017)
pas de stockage intermédiaire, F	Wang et al. (2010b)
$C_{max}, \sum w_i C_i$	Balaji and Porselvi (2014)
No-wait, L_{max}	Wang et al. (2010a)

TABLE 1.2: problèmes flow-shop

La table 1.3 énumère quelques travaux concernant les problèmes flowshop hybrides.

Critère d'optimalité	Référence
$HF2(0, PM^{(2)})$	
C_{max}	Chang et al. (2004), Gupta and Tunc (1994), Tseng et al. (2008)
$\sum C_j$	Guirchoun et al. (2005)
$HF2(0, RM^{(2)})$	
C_{max}	Riane et al. (2002)Riane et al. (1998)
$HF2(0, QM^{(2)})$	
C_{max}	Huang and Li (1998)
$HF2(PM^{(1)}, 0)$	
C_{max}	Gupta et al. (1997)
$HF2(RM^{(1)}, 0)$	
C_{max}	Figielska (2009), Low et al. (2008), Oguz et al. (1997)
$HF2(PM^{(1)}, PM^{(2)})$	
C_{max}	Dong et al. (2017), Haouari et al. (2006), Hidri et al. (2014), Lee and Vairaktarakis (1994), Wang et al. (2005), Xie and Wang (2005)
$C_{max} + \sum T_i$	Aurich et al. (2016)
$HF2(RM^{(1)}, RM^{(2)})$	
C_{max}	Rabiee et al. (2014), Suresh (1997)
$HF2(QM^{(1)}, QM^{(2)})$	
C_{max}	Dessouky et al. (1998), Kyparisis and Koulamas (2006a), Soewandi and Elmaghraby (2003)
$HFk(PM^{(1)}, PM^{(2)}, \ldots, PM^{(k)})$	
$\sum w_i C_i$	Kyparisis and Koulamas (2001), Shiau et al. (2008)

C_{max}	Negenman (2001), Paternina-Arboleda et al. (2008), Kurz and Askin (2004), Brockmann et al. (1998), Carlier and Neron (2000), Kahraman et al. (2008), Portmann et al. (1998),Engin and Doyen (2004), Zandieh et al. (2006), Alaykyran et al. (2007), Pan et al. (2014), Jin et al. (2006), Shi et al. (2015), Belkadi et al. (2006), Yi et al. (2016), Lei and Guo (2016), Yazdani and Naderi (2016), Djellab and Djellab (2002), Hidri and Haouari (2011), Sawik (2000), Wardono and Fathi (2004)
C_{max}, F	Pachedo et al. (2013)
$\sum C_i$	Azizoglu et al. (2001), Pan and Dong (2014)
C_{max}, T_{max}	Mousavi et al. (2013), Naderi and Sadeghi (2012)
$HFk(QM^{(1)}, \ldots, QM^{(k)})$	
C_{max}	Dessouky et al. (1998), Kyparisis and Koulamas (2006b), Verma and Dessouky (1999), Sevastianov (2002), Amin-Naseri and Beheshti-Nia (2009)
$HFk(RM^{(1)}, \ldots, RM^{(k)})$	
C_{max}	Chen et al. (2007), Yaurima et al. (2009), Mahdavi et al. (2011)

TABLE 1.3: Problèmes flowshop hybrides

Chapitre 2

Optimisation multi-critère

On donne dans ce qui suit certaines définitions sur la théorie de l'optimisation combinatoire et de la théorie de la complexité. Elles sont principalement puisées sur les ouvrages de (Marlow, 1993, Siarry, 2014). Un problème d'optimisation combinatoire consiste à trouver parmi un ensemble fini de possibilités, le meilleur choix. La notion de meilleur choix est déterminé par une fonction objectif.

2.1 Définition

Un problème d'optimisation est la donnée d'une fonction objectif $f : D \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ où D est un ensemble ouvert, appelé ensemble de recherche. Un programme mathématique est un problème d'optimisation sous contrainte où le but est la recherche d'un $x_0 \in D$ vérifiant :

$$f(x_0) = \min\{f(x) : x \in K\} \tag{2.1}$$

où K est spécifié. Le vecteur x est appelé vecteur de décision, x_0 vecteur solution et K l'ensemble des contraintes. On utilise l'ecriture suivante pour présenter le problème :

$$\text{minimiser} \quad f(x) \tag{2.2}$$

$$\text{s.c} \qquad x \in K \tag{2.3}$$

Lorsque $K = D$, le problème est dit sans contraintes. On appelle K l'ensemble des solutions réalisables ou encore domaine de définition de f. Ainsi, x est dite solution réalisable si et seulement si $x \in K$.

On peut remarquer qu'un problême d'optimisation peut être une maximisation ou une minimisation (il suffit de poser $f' = -f$).

Le domaine de définition peut être : vide (dans ce cas, le programme n'admet pas de solutions), dans le cas contraires, le programme admet des solutions.

On appelle extrêmum de f sur K tout minimum ou maximum du problème et on peut distinguer plusieurs types. On dit qu'il existe un minimum global en $x_0 \in K$ si $f(x) \geqslant f(x_0), \forall x \in K, x \neq x_0$ et si $f(x) > f(x_0), \forall x \in K, x \neq x_0$, il est appelé minimum global strict. On dit qu'il existe un minimum local en $x_0 \in K$ si $f(x) \geqslant f(x_0)$ est vraie pour tout $x \in V \cap K$, où $V = V(x_0)$ est un

voisinage de x_0 et si $f(x) > f(x_0), \forall x \in K \cap V$, le vecteur est appelé maximum global strict.
Si l'ensemble K est discret, on parle de programme mathématique discret. On l'appelle entier si $K \in \mathbb{N}^n$ ou binaire si $K \in \{0,1\}^n$. Si certaines composantes du vecteur x prennent des valeurs dans un ensemble discret et les autres dans un ensemble continue, on parle de programme mathématique mixte.
On peut diviser les problèmes d'optimisation sous contraintes en cinq sous-problèmes :

- Problème sous contraintes d'égalité, défini comme suit :

$$\begin{aligned} \text{minimiser} \quad & f(x) && (2.4)\\ \text{s.c} \quad & h(x) = 0 && (2.5) \end{aligned}$$

  où $h : D \longrightarrow \mathbb{R}^n$.
- Problème sous contraintes d'inégalité, présenté comme suit :

$$\begin{aligned} \text{minimiser} \quad & f(x) && (2.6)\\ \text{s.c} \quad & g(x) \leqslant 0 && (2.7) \end{aligned}$$

  où $g : D \longrightarrow \mathbb{R}^n$.
- Problème sous contraintes mixtes, présenté comme suit :

$$\begin{aligned} \text{minimiser} \quad & f(x) && (2.8)\\ \text{s.c} \quad & h(x) = 0 && (2.9)\\ & g(x) \leqslant 0 && (2.10) \end{aligned}$$

- Problème de programmation convexe, présenté comme suit :

$$\begin{aligned} \text{minimiser} \quad & f(x) && (2.11)\\ \text{s.c} \quad & g(x) \leqslant 0 && (2.12)\\ & Ax = b && (2.13) \end{aligned}$$

  où f et chaque fonction coordonnée de g est une fonction convexe. A est une matrice $k \times n$ et $b \in \mathbb{R}^k$.
- Programme linéaire. Il est présenté comme suit :

$$\begin{aligned} \text{minimiser} \quad & c^T x && (2.14)\\ \text{s.c} \quad & Ax = b && (2.15)\\ & x \geqslant 0 && (2.16) \end{aligned}$$

  où A est une matrice $m \times n$ satisfaisant certaines conditions, $b \in \mathbb{R}^m$ et $c \in \mathbb{R}^n$. La notation c^T représente la transposée du vecteur.

2.2 Optimisation multicritère

Dans la plupart des problèmes qu'on rencontre dans le monde réel, on cherche une solution qui optimiserait plusieurs critères à la fois. Les premières recherches concernant l'optimisation multicritère ont été menés au XIX-eme siècle dans le domaine de l'économie par Edegeworth et généralisés par Pareto.
Dans un problème d'optimisation multicritère, on cherche à optimiser plusieurs critères souvent contradictoires. Il est possible de ne pas trouver une solution satisfaisant toutes les contraintes et optimisant simultanément les critères considérés et dans ce cas, on cherche une solution représentant un compromis acceptable entre les différents objectifs.

Définition 2.1. Un problème d'optimisation multi-objectif est défini comme suit :

$$\text{minimiser} \quad f(x) = (f_1(x), f_2(x), \ldots, f_r(x)) \tag{2.17}$$

$$\text{s.c} \qquad x \in K \tag{2.18}$$

où $f_i : D_i \subset \mathbb{R}^n \longrightarrow \mathbb{R}$.

Définition 2.2. Soient deux solutions x, x'. On dit que x' domine x si et seulement $\forall i \in \{1, \ldots, r\}, f_i(x) \geqslant f_i(x')$ et $\exists j \in \{1, \ldots, r\}, f_j(x) > f_j(x')$.

On dit qu'une solution $x_0 \in K$ est Pareto optimal (ou efficace) si il n'existe aucun $x \in K$ tel que $f(x)$ domine $f(x')$.

On appelle front Pareto l'ensemble des solutions Pareto optimales omposé de points qui ne sont dominés par aucun autre point. Appelé aussi surface de compromis ou ensemble des solutions efficaces.

2.3 Méthode d'optimisation multicritère

Les approches de résolution des problèmes multiobjectifs peuvent être réparties en quatre classes :

- Les méthodes Métaheuristiques.
- Les méthodes non Pareto.
- Les méthodes Pareto.
- Les méthodes hybrides.

2.3.1 Méthodes métaheuristiques

Les métaheuristiques permettent de résoudre des problèmes ayant des instances de grandes tailles et sont utilisées dans plusieurs domaines tels que : la robotique, télécommunications, ordonnancement, etc. On peut les classer selon leur nature :

Inspirée par la nature ou non Plusieurs métaheuristiques sont inspirées par des systèmes naturels : Algorithmes évolutionnaires (génétiques), colonies de fourmis (biologie). Le recuit simulé (physique). Optimisation par essaim particulaire (sciences sociales).

Recherche basée sur une seule solution ou une population de solutions Les algorithmes basés sur une seule solution manipulent et transforment une seule solution durant la recherche tandis que dans les algorithmes basés sur une population, plusieurs solutions sont manipulées.

itérative ou gloutonne Dans un algorithme itératif, on démarre avec une population entière qu'on transforme à chaque itération utilisant certains opérateurs alors que dans un algorithme greedy, on démarre d'une solution nulle et à chaque étape, une variable de décision est ajoutée jusqu'à la complétude de la solution. La plupart des métaheuristiques sont itératives.

Quelques exemples de métaheuristiques :

Le recuit simulé Le recuit simulé a été introduit par Kirkpatrick et al. (1983) et est inspirée par le processus physique du recuit utilisé en metallurgie.

La recherche tabou La recherche tabou a été introduite par Glover (1986). Elle combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes pour surmonter l'obstacle des optima locaux.

Les algorithmes génétiques Les algorithmes génétiques ont été introduits par Holland (1984) et sont basées sur la théorie de l'évolution de Darwin et de la procréation selon les règles de Mendell.

2.3.2 Les méthodes non Pareto

Dans ces méthodes, on cherche à ramener le problème multiobjectif à un ou plusieurs problèmes mono-objectif. Citons quelques exemples :

Les méthodes agrégées Parmi les méthodes utilisant cette approche, on peut citer

- Agrégation par pondération : Cette méthode consiste à assigner à chaque fonction objectif $f_i, i \in \{1, \ldots, r\}$ un poids $w_i \in [0, 1]$ où $\sum_{i=1}^{r} w_i = 1$ et de considérer le nouveau problème :

$$\text{Minimiser} \quad \tilde{f}(x) = \sum_{i=1}^{r} w_i f_i(x) \tag{2.19}$$

$$\text{s.c} \quad x \in K \tag{2.20}$$

- Méthode ϵ-contrainte : Appelée aussi méthode du compromis. Transforme le problème en un problème mono-objectif en choisissant l'un des objectif comme objectif à optimiser prioritairement et les autres objectifs en contrainte d'inégalités par un vecteur contraintes ϵ comme suit :

$$\text{Minimiser} \quad f_1(x) \tag{2.21}$$

$$\text{s.c} \quad f_2(x) \leqslant \epsilon_2, f_3(x) \leqslant \epsilon_3, \ldots, f_r(x) \leqslant \epsilon_r \tag{2.22}$$

$$x \in K \tag{2.23}$$

La méthode lexicographique Méthode introduite par Fourman (1985) qui consiste à ranger les objectif par ordre d'importance établi par le décideur. Ensuite, l'optimum est obtenu en minimisant les fonctions objectifs en suivant l'ordre en transformant les fonctions objectifs déjà

minimisées en contraintes.

2.3.3 Les méthodes Pareto

Ces méthodes traitent les objectifs sans les transformer pendant la résolution. Voici quelques algorithmes utilisant cette méthode :

MOGA (Multi-Objective Genetic Algorithm) Introduit par Fonseca and Fleming (1993) et utilise la notion de dominance pour ranger les solutions.

NSGA-II (Non Sorting Genetic Algorithm II) Introduit par Deb et al. (2002).

SPEA2 (Strength pareto evolutionary Algorithm) Introduit par Zitzler et al. (2001).

2.3.4 Méthodes hybrides

La méthode hybride consiste à combiner plusieurs méthodes afin d'améliorer les performances d'un algorithme. Un cas particulier d'hybridation est la combinaison d'un algorithme génétique avec le principe de recherche locale. On peut citer les méthodes suivantes :

les méthodes MOTS combinant une population et une recherche Tabou,

Les méthodes PSA combinant un algorithme génétique et le recuit simulé,

Les méthodes M-PAES intégrant un schéma généralisant l'implémentation d'un grand nombre d'algorithmes hybrides pour l'optimisation multiobjectif.

2.3.5 Principe de quelques algorithmes

Le recuit simulé

Le recuit simulé est une méthode de recherche locale stochastique dont les étapes de l'algorithme sont décrits ci-dessous :

1– La donnée d'une température minimale T, une longueur de la période L, une fonction de refroidissement dépendante de la température en cours. Choix d'une température initiale $T_1 > 0$ et établir le compteur à $k = 1$.

2– Génération d'une solution initiale S. Calculer son "coût" $C(S)$.

3– Répéter les étapes suivantes L fois.

   1– Créer une solution S' à partir de S et évaluer son coût.

   2– Calculer $\delta = C(S') - C(S)$.

   3– Si $\delta < 0$, remplacer S par S'.

   4– Si $\delta \geqslant 0$ choisir S' à la place de S avec une probabilité $e^{-\frac{\delta}{T_k}}$.

4– Si le critère d'arrêt n'est pas atteint, calculer la nouvelle température $T_{k+1} = F(T_k)$ et poser $k = k + 1$, revenir à l'étape 3.

Les algorithmes génétiques

L'algorithme peut être divisé en plusieurs étapes principales.

1– Génération aléatoire d'une population initiale contenant N solutions réalisables, appelées individus.

2– Évaluation de la qualité de chaque individu permettant de définir une fonction de mérite (fitness).

3– Sélection de deux parents selon l'une des méthodes suivantes :

Sélection par roulette : Cette méthode s'inspire des roues de la loterie. On associe à chaque individu une probabilité de sélection proportionnelle à la valeur de sa fonction de mérite.

Sélection par rang : Dans cette technique, les individus sont triés selon leur valeur de la fonction fitness. Ensuite, un rang proportionnel est attribué à chaque individu proportionnellement à sa valeur de fonction de mérite. On choisit les individus ayant les meilleurs rangs.

Sélection par tournoi : Cette technique consiste à comparer plusieurs individus pris au hasard et de choisir le meilleur individu de par la valeur de la fonction de mérite. On peut décider de choisir plus d'une vainqueur selon le nombre de tournois qu'on souhaite établir. Un même individu peut participer à plus d'un tournoi.

Élitisme : Cette méthode permet de mettre en avant les meilleurs individus de la population.

4– Croisement des parents. On peut décider de faire des croisements aléatoire mais la méthode la plus couramment utilisée est le croisement multi-point qui consiste à découper chaque individu choisi pour le croisement en N morceaux (2 ou plus) puis on prend un gène de chaque individu pour créer un autre individu. En prenant X individus, on peut créer X nouveaux individus. On vérifiera que chaque individu créé est une solution réalisable.

5– mutation des parents qui consiste à modifier dans l'individu choisi un gène qui permettra de créer un nouvel individu qui représente une solution réalisable.

6– Répéter les étapes 3,4 et 5 jusqu'à obtention d'une population ayant N individus.

7– Itérer l'algorithme à partir de l'étape 2 jusqu'à ce que la condition d'arrêt soit satisfaite.

2.4 Types de problèmes

2.4.1 Complexité algorithmique

Tout algorithme a besoin des ressources temps et espace pour résoudre un problème donné. La complexité d'un algorithme représente le nombre d'étapes nécessaires pour résoudre un problème de taille n.

Le but du calcul de la complexité algorithmique est d'obtenir une borne asymptotique du nombre d'étapes. La notation la plus utilisée pour représenter l'analyse asymptotique est O.

Définition 2.3. On dit qu'un algorithme est de complexité $f(n) = O(g(n))$ s'il existe $n_0 > 0$, $c > 0$ tels que $\forall n \geqslant n_0$, $f(n) \leqslant c.g(n)$.

La fonction f est donc limitée supérieurement par g. Cette notation peut être utilisée pur calculer la complexité par rapport au temps ou l'espace.

Définition 2.4 (Temps polynomial). Un algorithme est dit polynomial si sa complexité est $O(p(n))$, où p est un polynôme de degré n.

Définition 2.5 (Temps exponentiel). Un algorithme est dit exponentiel si sa complexité est $O(c^n)$, où c est un nombre réel supérieur strictement à 1.

La complexité d'un problème est équivalente à la complexité du meilleur algorithme résolvant ce problème.

La théorie de la complexité s'intéresse aux problèmes de décision.

Définition 2.6. Un problème est dit de décision si la réponse est oui ou non.

Tout problème d'optimisation peut être réduit en un problème de décision.

Les problèmes de décision sont divisées en deux classes importantes :

La classe P qui représente l'ensemble de tous les problèmes de décision qui peuvent être résolues par un algorithme polynomial déterministe.
Un algorithme est dit polynomial pour un problème de décision A si la complexité empirique est majorée par une fonction polynomiale $p(n)$.
Ainsi la classe P représente la famille de problèmes où il existe un algorithme en temps polynomial résolvant ce problème.

La classe NP qui représente l'ensemble de tous les problèmes qui peuvent être résolues par un algorithme non déterministe en temps polynomial.

Définition 2.7. On dit qu'un problème de décision B est réductible en temps polynomial (polynomially reduced) en un problème de décision A s'il existe un algorithme polynomial pouvant convertir chaque instance I_B de B en une instance I_A de A.

Définition 2.8. On dit qu'un problème A est NP-difficile (NP-Hard) si chaque autre problème NP est réductible en temps polynomial en A. Il est NP-complet s'il est dans la classe NP et que tout autre problème dans NP est réductible en temps polynomial en A.

2.4.2 Algorithmes approchées

Définition 2.9 (Algorithme ϵ-approché). On dit qu'un algorithme A approche l'optimum d'un facteur ϵ si sa complexité et polynomiale et pour chaque instance I, il produit une solution A(I) qui vérifie :

$$\frac{A(I)}{Opt(I)} \leqslant \epsilon$$

Définition 2.10 (Schéma d'approximation en temps polynomial). Un algorithme est dans la classe PTAS (polynomial time approximation scheme) s'il admet un algorithme $(1+\epsilon)$-polynomial pour tout ϵ fixé.

Le problème du voyageur de commerce euclidien appartient à cette classe.

Définition 2.11 (Schéma d'approximation entièrement en temps polynomial). Un algorithme est dans la classe FPTAS (fully polynomial time approximation scheme) s'il admet un algorithme $(1+\epsilon)$-polynomial en terme de taille d'instance du problème et $\frac{1}{\epsilon}$-polynomial pour tout ϵ fixé.

Le problème du sac à dos appartient à cette classe.

Chapitre 3

Tâches déteriorables : Etat de l'art

Dans les problèmes d'ordonnancement classiques, le temps d'exécution est considéré fixe. Mais dans la réalité , il existe certains modèles où le temps de traîtement d'une tâche peut varier selon certains facteurs tel que le temps. On peut citer par exemple le temps de traitement des laminoirs d'acier où le temps nécessaire à la fabrication augmente avec la baisse de la température ou en agriculture où la moisson diminue avec la baisse de la récolte. De telles tâches sont appelées tâches déteriorables.

Dans la section suivante, on présente un état de l'art des recherches effectuées sur les problèmes d'ordonnancement considérant des tâches déteriorables.

3.1 Etat de l'art

Le problème d'ordonnancement de tâches déteriorables (DJSP : deteriorating jobs scheduling problem) a été introduit par Browne and Yechiali (1990), dans lequel on considère un modèle stochastique où on doit ordonnancer un ensemble de n tâches à temps d'exécution variable dont l'objectif est de déterminer un ordonnancement qui minimise le temps d'exécution total (makespan). Dans ce cas, le makespan n'est plus un résultat indépendant de la règle d'ordonnancement utilisée lorsque la préemption et le temps d'attente ne sont pas autorisés. Ce problème a été traité indépendamment par Gupta and Gupta (1988), Tanaev et al. (1994).

Dans ces problèmes, on considère que le temps d'exécution est une fonction linéaire du temps donnée par l'équation $p_i = p_i^0 + \alpha_i t_i$, où $p_i, p_i^0, \alpha_i, t_i$ représentent respectivement le temps d'exécution actuel, le temps d'exécution initial, le taux de déterioration et le début du temps d'exécution de la tâche T_i.

3.1.1 Une seule machine

Alidaee and Landram (1996), Gawiejnowicz and Pankowska (1995) ont proposé des heuristiques pour le problème $1|p_i^0 + \alpha_i t|C_{max}$. Cheng and Ding (1998b) ont prouvé que le problème est NP-difficile lorsqu'il existe des dates d'arrivée, même pour $p_i^0 = 1$. Un algorithme Branch and Bound a été proposé par Lee et al. (2008). Cheng and Ding (2000) ont prouvé que les problèmes $1|p_i^0 + \alpha_i t, d_i|F_{max}$ et $1|p_i^0 + \alpha_i t, d_i|C_{max}$ sont équivalents.

En considérant des contraintes de précédence, Wang et al. (2008) ont prouvé qu'il existe un ordonnancement optimal pour le problème de minimisation du makespan, lorsque la relation de précédence est sous forme de chaînes et présentent un algorithme polynomial lorsqu'elle est sous forme de graphe séries parallèles.

En considérant le fait que lorsque le nombre de tâches augmente, le temps de début de leur exécution par la machine sera retardée, rendant ainsi le temps d'exécution basique moins pertinent, Mosheiov (1994) a proposé de simplifier l'expression de la fonction de temps d'exécution et a donc étudié le problème $1|\alpha_i t, r_i|C_{max}$. Gawiejnowicz (2007) a prouvé que le problème $1|\alpha_i t, r_i, d_i|C_{max}$ est NP-complet au sens fort lorsque r_i, d_i sont arbitraires et NP-complet lorsque $r_i \in \{r_1, r_2\}, d_i \in \{d_1, d_2\}$. Lee and Wu (2003) proposent un algorithme de programmation en nombre entiers pour le problème $1|\alpha_i t, ma, resumable|C_{max}$, où $[a_1, a_2]$ est la période d'indisponibilité de la machine.

Gawiejnowicz (2007) a prouvé que le problème $1|\alpha_i t, ma, k, non-resumable|C_{max}$, où k désigne le nombre de périodes de maintenance est NP-complet lorsque $k = 1$ et NP-complet lorsque k est arbitraire. Ji et al. (2006) ont présenté un PTAS. Ils présentent aussi un algorithme en-ligne de ratio b_1/t_0, où b_1 représente le début de la maintenance, et un FPTAS dans le cas hors-ligne en $O(\frac{n^2}{\epsilon})$ pour le cas $k = 1$.

Mosheiov (1991) montre que l'ordonnancement optimal pour le problème $1|p_i^0 + \alpha_i t|F$ est V-shaped[1] selon α_i.

Li et al. (2009) ont prouvé que l'ordonnancement optimal avait une propriété V-shaped selon a_i et proposent 2 heuristiques pour le problème $1|a_i(\alpha + \beta t_i)| \sum_{i=1}^{n} \sum_{j=1}^{n} |C_j - C_i|$.

Les problèmes $1|p_i^0+\alpha_i t, w_i|F_w$ et $1|p_i^0+\alpha_i t, r_i, d_i, w_i|F_w$, $1|p_i^0+\alpha_i t, d_i|L_{max}$ sont NP-difficiles (Bachman and Janiak, 2000, Bachman et al., 2002, Cheng et al., 2004). Le problème $1|p_i^0 + \alpha_i t, r_i, d|N_T$ est polynomial (Cheng et al., 2004).

Cheng and Ding (1998a) ont prouvé que les problèmes $1|p_i^0 - \alpha_i t, d_i|C_{max}$, $1|p_i^0 - \alpha_i t, d_i|L_{max}$ et $1|p_i^0 - \alpha_i t, d_i|F_{max}$, sont NP-difficiles dans les cas où $d_i \in \{d_1, d_2\}, \alpha_i = \alpha$ et $\alpha_i d_i < p_i^0 < d_i$. Cheng and Ding (1998b) ont prouvé que le problème $1|p_i^0 - \alpha_i t, r_i|C_{max}$ se résout polynomialement dans les cas suivants : $\alpha_i = \alpha$, $p_i = 1 - \alpha_i t_i$ et qu'il est NP-difficile au sens fort pour des dates de disponibilité arbitraires et au moins NP-difficile lorsque $r_i \in \{0, R\}$.

Wang (2009b) présente des algorithmes polynomiaux pour le problème $1|p_i^0 - \alpha_i t, prec|C_{max}$ dans le cas où la relation de précédence représente des chaînes parallèles ou un graphe en séries parallèles.

Ng et al. (2002) montrent que le problème $1|p_i^0 - \alpha_i t|F_{max}$ admet un ordonnancement optimal en temps polynomial pour le cas où $\alpha_i = \alpha$ et $\alpha_i = ka_i$ et montrent que dans le cas général, tout ordonnancement optimal doit respecter la propriété /\-shaped selon le taux de détérioration et la tâche de plus petit α_i est exécutée en premier et présentent un algorithme en programmation dynamique.

Wang et al. (2008) ont prouvé que le problème $1|p_i^0 - \alpha_i t, w_i, séries-parallèles|F_w$ peut être résolu en temps polynomial. Cheng et al. (2004) ont prouvé que le problème $1|p_i^0 - \alpha_i t, r_i, w_i, d|F_w$

1. ordonnancement V-shaped selon α_i : les tâches sont ordonnées selon l'ordre décroissant de leur taux de détérioration si elles sont placées avant la tâche dont le taux de détérioration est minimal et dans l'ordre croissant si après.

est NP-difficile.

Wang and Xia (2005) ont prouvé que le problème $1|p_i^0 - \alpha_i t, d_i|L_{max}$ est résoluble par la règle EDD et que le problème $1|p_i^0 - \alpha_i t, d_i|N_T$ admet un ordonnancement optimal lorsque $\alpha_i = ka_i$ et Woeginger (1995) a prouvé qu'il est résoluble polynomialement lorsque $d_i = d$. En supposant que $\alpha_i d_i < \alpha_i < d_i$, Ho et al. (1993) prouvent que le problème $1|p_i^0 - \alpha_i t, d_i|N_T$ est NP-difficile au sens fort lorsque les dates échues sont arbitraires et lorsqu'il y a deux dates échues distinctes, le problème est NP-difficile et pour $d_i = d$, le problème est résoluble en temps polynomial $O(n \log n)$. Lee and Wu (2003) proposent un algorithme de programmation en nombre entiers pour le problème $1|\alpha_i t, ma, resumable|C_{max}$ tandis que Ji et al. (2006) ont montré que le problème $1|\alpha_i t, ma, non - resumable|C_{max}$ est NP-difficile et ont présenté un algorithme approché en temps pseudo polynomial. Ils présentent aussi un algorithme en-ligne et un FPTAS dans le cas hors-ligne. Gawiejnowicz (2007) a prouvé que le problème $1|\alpha_i t, ma, non - resumable, k|C_{max}$, où k est le nombre de périodes de maintenance, est NP-difficile lorsque $k = 1$ et NP-difficile au sens fort lorsque k est arbitraire.

Mosheiov (1994) a montré que le problème $1|\alpha_i t|F_{max}$ se résout en ordonnant les tâches selon l'ordre croissant de leur taux de détérioration (SDRF). Ji et al. (2006) ont montré que le problème $1|\alpha_i t, ma, non - resumable|F_{max}$ est NP-difficile et ont présenté un algorithme approché en temps pseudo polynomial et Fan et al. (2011) ont prouvé que le problème $1|\alpha_i t, ma, resumable, k|F_{max}$ ont proposé un FPTAS pour $k = 1$ et ont prouvé qu'il n'y avait aucun FPTAS avec un ratio constant lorsque $k > 1$.

Oron (2008) a prouvé que, pour le problème $1|\alpha_i t| \sum_{i=1}^{n} \sum_{j=i}^{n} |C_j - C_i|$, l'ordonnancement optimal avait une propriété V-shaped selon α_i et proposent 2 heuristiques en temps polynomial. Mosheiov (1994) a montré que l'ordonnancement des tâches pour le problème $1|\alpha_i t, w_i| \sum_{i=1}^{n} w_i C_i$ se fait selon l'ordre croissant des ratio $\alpha_i/[(1 + \alpha_i)w_i]$ et Cheng et al. (2004) ont prouvé que le problème $1|\alpha_i t, r_i, d_i, w_i| \sum_{i=1}^{n} w_i C_i$ se résout polynomialement et vérifie la propriété $\bigwedge$-shaped. Wang et al. (2011a) ont prouvé que le problème $1|\alpha_i t, chains| \sum_{i=1}^{n} C_i^2$ se résout polynomialement.

Mosheiov (1994) a proposé le principe d'ordonnancement SDRL pour le problème $1|\alpha_i t, d_i|T_{max}$ EDD pour le problème $1|\alpha_i t, d_i|L_{max}$. Mosheiov (1994) a proposé un algorithme MINLID (minimizing number of tardy jobs under linear deterioration) qui donne une solution optimale pour le problème $1|\alpha_i t, d_i|N_T$ et un algorithme dynamique polynomial lorsque $d_i = d$.

Dans d'autres études, on considère l'existence d'une ou plusieurs dates de début de détérioration et ce problème est formulé par une fonction de déterioration par morceaux. En supposant qu'il existe une seule date de début de déterioration, (Mosheiov, 1995) prouve que le problème est NP-difficile et présente une heuristique qu'il adapte au cas de l'existence de plusieurs dates de déterioration.

Kubiak and van de Velde (1998) montrent que le problème $1|p_i|C_{max}$ est NP-difficile et présentent un algorithme pseudo-polynomiaux et un FPTAS est donné par Kovalyov and Kubiak (1998) où la fonction de détérioration est donnée par :

$$p_i(t) = \begin{cases} p_i^0 & si\ t \leq d \\ p_i^0 + \alpha_i(t - d) & si\ d < t < D \\ p_i^0 + \alpha_i(D - d) & si\ t \geq D \end{cases} \tag{3.1}$$

où d, D sont les dates de début et fin de déterioration.

En considérant la fonction suivante :

$$p_i(t) = \begin{cases} p_i^0 & si\ t_i \leq d_i \\ p_i^0 + \beta_i & si\ t_i > d_i \end{cases} \tag{3.2}$$

Sundararaghavan and Kunnathur (1994) ont proposé une heuristique optimale pour le problème $1|p_i|C_w$ pour le cas où $d_i = d$ et $p_i^0 = p^0$ et un algorithme non optimal pour le cas où $w_i \geq w_j \Rightarrow \beta_i \geq \beta_j$. Guo et al. (2014) montrent que le problème $1|p_i|T$ est NP-difficile et présentent une heuristique appelée General variable neighborhood search (GVNS) et une autre appelée simple weighted search procedure (SWSP).

En considérant la fonction suivante :

$$p_i(t) = \begin{cases} a_i & si\ t \leq d \\ a_i - \alpha_i(t-d) & si\ d < t < D \\ a_i - \alpha_i(D-d) & si\ t \geq D \end{cases} \tag{3.3}$$

où $0 < \alpha_i < 1$ et $a_i > \alpha_i(\min\{\sum_{j=1,j\neq i}^{n} a_j, D\} - d)$, Cheng et al. (2003) montrent que les problèmes $1|p_i|C_{max}$ et $1|p_i|F_{max}$ sont NP-difficile dans le cas $d = 0, D < \infty$ et un FPTAS est proposé par Ji and Cheng (2007) pour résoudre le problème $1|p_i|C_{max}$.
Lorsque $d > 0$, Cheng et al. (2003) prouvent que le problème $1|p_i|C_{max}$ est NP-difficile et présentent un algorithme pseudo-polynomial.

En considérant la fonction suivante :

$$p_i = \begin{cases} p_i^0, & t_i \leq d \\ p_i^0 + \alpha_i t_i, & t_i > d \end{cases} \tag{3.4}$$

Alidaee and Womer (1999) ont prouvé que le problème $1|p_i|C_{max}$ est polynomial lorsque $p_i^0 = a$.

Alidaee and Womer (1999) ont prouvé que les problèmes $1|p_i = p_i^0 + \alpha_i t_i^r|C_{max}$ et $1|p_i = p_i^0 + \alpha_i t_i^r|C_{max}$ sont polynomiaux.

En considérant la fonction suivante

$$p_i = \begin{cases} a_i, & t_i \leq d_i \\ g(t_i) & t_i > d_i \end{cases} \tag{3.5}$$

où g_i est une fonction croissante, Alidaee and Womer (1999) ont proposé un algorithme en modifiant celui de Moore pour résoudre le problème $1|p_i|Nb$.

Cheng et al. (2004) ont prouvé que le problème $1|p_i = \max\{p_i^0, p_i^0 + \alpha_i(t_i - d)\}|F_w$ est NP-difficile. Kunnathur and Gupta (1990) présentent un algorithme Branch and bound et un algorithme de programmation dynamique et une heuristique pour résoudre le problème $1|p_i = p_i^0 + \max\{0, \alpha_i(t - d_i)\}|C_{max}$. Mosheiov (2005) a prouvé que le problème $1|p_i = p_i^0 r^a|F$, où r représente la position de la tâche dans l'ordonnancement, admet un ordonnancement optimal qui respecte la propriété V-shaped selon p_i^0. En considérant le problème $1|p_i = p_i^0 + \alpha_i f(t_i)|C_{max}$, où f est une fonction positive croissante, Cai et al. (1998) donnent des algorithmes optimaux dans les 2 cas suivants :

$p_i^0 = p^0$ et $\alpha_i = \alpha$ et présentent un FPTAS dans le cas où $f(d,t) = \max\{0,(t_i - d)\}$. Pour $p_i^0 = 1$, Mel'nikov and Shafransky (1980) ont prouvé que l'ordonnancement optimal est obtenu en arrangeant les tâches selon l'ordre décroissant des p_i^0 même lorsque la fonction est décroissante dérivable et sa dérivée est inférieure à 1 en valeur absolue.
En considérant le problème $1|p_i = p_{ir} = \alpha_i t_i r^{\beta_i}, d|C_{max}$, Yang and Kuo (2011) ont présenté un algorithme polynomial en temps $O(n^3)$ pour le problème le problème $1|p_i = p_{ir} = \alpha_i t_i r^{\beta_i}, d|C_{max}$ et un algorithme polynomial en temps $O(n^3 \log n)$ pour le problème le problème $1|p_i = p_{ir} = \alpha_i t_i r^{\beta_i}, d|Nb$.

D'autres auteurs ont étudié le problème d'ordonnancement sur une machine avec tâches déteriorables pour des cas particuliers : $p_{ir} = a_i(\alpha a^{r-1} + \beta)(bt + c)$ (Wang et al., 2009), $p_{ir} = a_i\alpha(t)(M + (1-M)r^a)$ où $0 < M < 1$ (Wang, 2009a).

En supposant que $p_i < p_j \Rightarrow w_i \geq w_j$, Wu et al. (2011) ont prouvé que le problème $1|p_i|F_w$ se résout optimalement selon l'ordre croissant des $\frac{p_i}{w_i}$. lorsque $p_i = p$, Wu et al. (2011) ont prouvé qu'on obtient une solution optimle en ordonnançant les tâches selon l'ordre décroissant des w_i.

Grande tardivité : Chaque tâche est définie par une date échue d_i.

En supposant que $p_i < p_j \Rightarrow d_i \leq d_j$, Wu et al. (2011) ont prouvé qu'on obtient une solution optimale en ordonnançant les tâches selon l'ordre croissant des d_i pour le problème $1|p_i, d_i|T_{max}$ et ont obtenu le même résultat dans les deux cas suivants, $p_i = p$ et $d_i = kp_i$.
En considérant le problème $1|p_{ir} = p_i^0(\alpha(t) + f(r))|C_{max}$, où f est une fonction décroissante de $[1,\infty[$ vers $]0,1]$ avec $f(1) = 1$, Wu et al. (2011) (2011) ont prouvé qu'un ordonnancement optimal est obtenu en triant les tâches selon l'ordre croissant des p_i^0 lorsque la fonction α est une fonction convexe dérivable. En considérant le problème $1|p_{ir} = (p_0 + \alpha_i t_i)r^a|C_{max}$,où $a < 0$, Wang and Cheng (2007) proposent un algorithme optimal lorsque $a \leq -1$ et dans le cas où α_i, a vérifient les conditions suivantes : $\alpha_1 \geq \max_{i=\overline{2,n}}\{\alpha_i\},\ \alpha_3 \geq 3^{-a}\beta,\ \alpha_{i+1} - \alpha_i \geq [(i+1)^{-a} - i^{-a}]\beta$ où $\beta > 0,\ i = \overline{3, n-1},\ \alpha_2 \geq \alpha_i,$ pour $i^{-a} \geq 2^{-a} + 1 + \frac{1}{\beta}, -1 < a < \frac{-4}{4+\beta}$. Ils prouvent que l'ordonnancement optimal est obtenu selon l'ordre croissant des α_i lorsque α_i, a vérifient les conditions suivantes : $\alpha_1 \geq \max_{i=\overline{2,n}}\{\alpha_i\}$ et $\max_{i=\overline{2,n}}\{\alpha_i\} \leq \frac{2^{-a}}{n}$ et $\frac{-1}{en} \leq a$.

3.1.2 Machines parallèles

Hsieh and Bricker (1997) proposent des heuristiques pour le problème $P_m|p_i^0 + \alpha_{ij}t|C_{max}$ et Kuo and Yang (2008) présentent un algorithme polynomial pour le problème $P_m|p_{ij} = p_{ij}^0 + \alpha_{ij}t_{ij}|F$ et un FPTAS a été présenté par Li and Yuan (2010) pour le problème $P_m|p_{ij} = p_{ij}^0 + \alpha_{ij}t_{ij}|C_{max} + \sum e_i$ et Hsu et al. (2013) ont proposé un algorithme polynomial pour le problème $P_m|p_{ij} = p_{ij}^0 + \alpha_j t_{ij}, ma, non-resumable|F$ et dans le cas $P_m|p_{ijr} = p_{ij}^0 + \alpha_j r, ma, non-resumable|F$. Kuo and Yang (2008) présentent un algorithme polynomial pour le problème $P_m|p_{ij} = p_{ij} = p_{ij}^0 - \alpha_{ij}t_{ij}|F_{max}$ avec $0 < \alpha_{ij} < 1, \alpha_{ij}(\sum_{l=1,l\neq i}^{n} a_{lj}) < a_{ij}$.

Mosheiov (1998) a prouvé que le problème $P_m|\alpha_{ij}t|C_{max}$ est NP-difficile même pour $m = 2$, Ji and Cheng (2009) prouvent que le problème $P|\alpha_{ij}t|C_{max}$, où le nombre de machines est arbitraire, est NP-difficile au sens fort lorsque le nombre est arbitraire et qu'aucun algorithme polynomial approché n'existe. Ren and Kang (2007) ont proposé un FPTAS pour le cas où $m = 2$ et le

généralisent pour le cas où le nombre de machines est m. Lee and Wu (2008) montrent que le problème $P_m|\alpha_{ij}t, ma|C_{max}$ avec une seule période de maintenance, est NP-difficile et proposent des heuristiques.

Chen (1996), Ji and Cheng (2009) montrent que le problème $P|p_{ij} = \alpha_{ij}t_{ij}|F$ est NP-difficile lorsque le nombre de machines est fixe et NP-difficile au sens fort lorsque le nombre est arbitraire, des heuristiques ont été proposées par Chen (1996). Li and Yuan (2010) ont présenté un FPTAS pour le cas $P_m|p_{ij} = \alpha_{ij}t_{ij}|\sum w_iC_i + \sum e_i$.

En considérant des fonctions où le temps d'exécution change en fonction d'une ou plusieurs dates à laquelle le temps d'exécution augmente, Mosheiov (1995) a prouvé que le problème $P_m|p_{ij}|C_{max}$ est NP-difficile et présente une heuristique lorsqu'il y a une seule date de début déterioration.

Plusieurs fonctions ont été étudiées.

En considérant la fonction suivante :

$$p_{ij}(t) = \begin{cases} p_{ij}^0 & si\ t_{ij} \leq d \\ p_{ij}^0 = -\alpha_{ij}(t_{ij} - d) & si\ d < t_{ij} < D \\ p_{ij}^0 - \alpha_{ij}(D - d) & si\ t_{ij} \geq D \end{cases} \tag{3.6}$$

où $0 < \alpha_{ij} < 1$ et $a_{ij} > \alpha_{ij}(\min\{\sum_{l=1,l\neq i}^{n} a_{lj}, D\} - d)$, Ji and Cheng (2007) ont présenté un FPTAS pour le problème $P_m|p_{ij}|C_{max}$ qui est une généralisation du FPTAS pour $m = 1$. Cheng et al. (2003) prouvent que les problèmes $R|p_{ij}|F_{max}$ et $Q|p_{ij}|F_{max}$ sont polynomiaux et que le problème $P|p_{ij}|C_{max}$ est NP-difficile lorsque $m = 2$ et NP-difficile au sens fort lorsque le nombre est arbitraire dans le cas où $D = \infty$, $d = 0$. Hsu et al. (2013) ont proposé un algorithme polynomial pour le problème $P_m|p_{ijr} = a_{ij}r^{\alpha_j}|F$.

Ji et al. (2016) ont proposé un FPTAS pour le problème $P_m|p_{ij} = a_{ij}[M+(1-M)r^{\beta}]+\alpha t_{ij}|C_{max}$, où $0 \leq M \leq 1, \beta \leq 0, \alpha > 0$ et ont prouvé que le problème $P_m|p_{ij} = a_{ij}[M+(1-M)r^{\beta}]+\alpha t_{ij}|F_{max}$ est polynomial.

3.1.3 Machines spécialisées

Pour le problème $F_m|p_i^0 + \alpha_{ij}t|C_{max}$, un algorithme branch and bound est proposé par Jafari et al. (2016) pour le cas $m = 2, \alpha_{ij} = \alpha_i$ et par Wang and Wang (2013) pour le cas $m = 3, \alpha_{ij} = \alpha$. Mosheiov (2002) montre que les problèmes $F_m|\alpha_{ij}t|C_{max}$ sont NP-difficiles pour $m \geq 3$ et présente des heuristiques pour chaque cas et montre que $J_m|\alpha_{ij}t|C_{max}$ est NP-difficile même pour $m = 2$. Zhao and Tang (2012) montrent que le problème $F_2|\alpha_{ij}t, prec|C_{max}$ est NP-difficile au sens fort lorsque la relation est de type 2[2] et présentent un algorithme optimal pour le cas où la relation de précédence est de type 1 [3]. Wang and Xia (2006) proposent des heuristiques pour le problème $F_m|\alpha_{ij}t, dominance^4|C_{max}$.

2. Une relation de précédence est de type 2 si lorsqu'une tâche T_i précède T_j, l'opération O_{j1} ne peut être exécutée sur M_1 que lorsque O_{i2} est achevée sur M_2.

3. On dit qu'une relation de précédence est de type 1 si lorsqu'une tâche i précède une tâche j, l'opération O_{jk} ne peut commencer avant la fin d'exécution de O_{ik} pour toute machine M_k.

4. On dit qu'une machine M_k domine la machine M_j et on écrit $M_j < M_k$ si et seulement si $\max_{i=\overline{1,n}}\{\alpha_{ij}\} \leq \min_{i=\overline{1,n}}\{\alpha_{ik}\}$.

Wang et al. (2006) ont montré l'existence d'un ordonnancement optimal pour le problème $F_2|p_{ij} = \alpha_{ij}t|F_{max}$ et présentent des algorithmes optimaux dans les cas où $a_{i2} = a_2$, la machine M_2 (resp. M_1 domine M_1 (resp. M_2 domine M_1) et un algorithme branch and bound lorsque $\alpha_{i1} = \alpha_{i2}$.

Cheng et al. (2015) ont proposé un algorithme branch and bound et une heuristique GHA (algorithme glouton) pour le problème $F_2|p_{ij} = \alpha_{ij}t, t_0 = 1|\lambda C_{max} + (1-\lambda)\sum_{i=1}^{n} C_i$. Les auteurs ont prouvé que dans le cas où $\alpha_{i2} = \alpha_2$, l'ordonnancement optimal est obtenu selon l'ordre croissant des α_{i1}, pour le cas où $\alpha_{i1} = \alpha_1$, l'ordonnancement optimal est obtenu selon l'ordre croissant des α_{i2} si $\alpha_1 \leq \min_{i=1,n} \alpha_{i2}$, selon leur ordre décroissant si $\alpha_1 \geq \max_{i=1,n} \alpha_{i2}$ et sii $\alpha_{i1} = \alpha_{i2}$, l'ordonnancement optimal est obtenu selon l'ordre croissant des α_{i1} ou α_{i2}. Wang and Xia (2006) ont considéré les cas $F_m|p_{ij} = \alpha_{ij}t, w_i, dominance, no-wait|F_w$, $F_m|p_{ij} = \alpha_{ij}t, w_i, dominance, no-idle|F_w$, $F_m|p_{ij} = \alpha_{ij}t, w_i, dominance, no-wait|L_{max}$ et $F_m|p_{ij} = \alpha_{ij}t, w_i, dominance, no-idle|L_{max}$ et ont proposé des algorithme optimaux pour différents cas de dominance.

Wang et al. (2006) ont considéré le problème $F_m|p_{ij} = p_{ij}^0 - \alpha_{ij}t|C_{max}$, où $\alpha_{ij} \in]0,1[$ et $\alpha_{ij}(\sum_{l\neq i} a_{lj}) < a_{ij}$. Les auteurs ont montré qu'un ordonnancement optimal peut être obtenu selon la règle de Johnson pour le problème lorsque $m = 2$ et $\alpha_{ij} = kp_{ij}^0$ et que l'ordonnancement optimal ne dépendait pas de la séquence lorque $m > 2$ et $p_{ij} = p_i^0(1-kt)$. Les auteurs ont aussi montré que le problème $F_2|p_{ij} = p_i^0(1-kt_i)|F_{max}$ admettaient un ordonnancement optimal obtenu en exécutant les tâches selon l'ordre croissant des p_i^0 et que le problème $F_2|p_{ij} = p_i^0(1-kt_i), d_i|L_{max}$ admet un ordonnancement optimal selon la règle EDD. Wang et al. (2006) ont montré qu'il existe un ordonnancement optimal pour le problème $F_m|p_{ij} = p_{ij}^0 - \alpha_{ij}t, d_i|N_t$. Wang et al. (2011b) ont étudié le cas $F_m|p_{ij} = p_{ij}^0(1-bt), dominance|C_{max}$ et ont proposé des algorithmes basés sur la règle de Johnson pour les cas de dominance suivants : $M_1 > M_2 > \ldots > M_{m-1}$ et $M_m > M_{m-1} > \ldots > M_2$ et des algorithmes polynomiaux dont la complexité dépend du nombre de machines pour les cas suivants : $M_2 > M_3 > \ldots > M_m$, $M_{m-1} > M_{m-2} > \ldots > M_1$, $M_2 < \ldots < M_h > \ldots > M_m$. Lee et al. (2014) ont prouvé que le problème $F_m|p_{ij} = p_{ij}^0 + \alpha_j t, d_i|L_{max}$ est NP-difficile et ont proposé un algorithme branch and bound ainsi que deux heuristiques de recuit simulé et essaim particulaire. Ng et al. (2010) ont proposé un algorithme branch and bound pour le problème $F_2|p_{ij}^0(\alpha+\beta t)|F_w$ et Bank et al. (2012) ont proposé une heuristique et un algorithme branch and bound pour résoudre le problème $F_2|p_{ij}^0(\alpha+\beta t), d_i|L_{max}$.

Deuxième partie

Ordonnancement et Gestion de l'Énergie

Chapitre 4

Ordonnancement et Gestion de l'Énergie : État de l'art

4.1 Introduction

Au cours du 20ème siècle, La population mondiale est passée de 1.65 milliards à 6 milliards. La révolution industrielle et les découvertes scientifiques ont permis cette évolution de la démographie en améliorant les conditions de vie. De ce fait, la population qui était à 3 milliards en 1960 à atteint 7 milliards en 2012 et l'ONU prédit que la population devrait atteindre les 8 milliards en 2023. La consommation énergétique suit l'évolution démographique et est donc en constante croissance et continuera à croître suivant la croissance démographique. L'évolution de la consommation énergétique est illustré dans la figure 4.1.

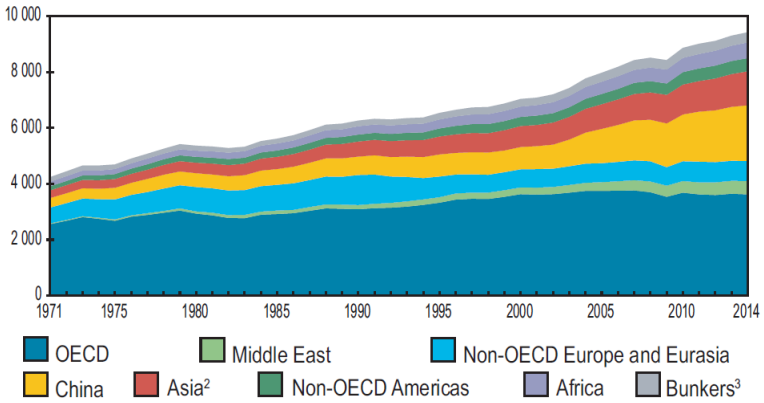


FIGURE 4.1 – Consommation d'énergie mondiale par région de 1971 à 2014 (International Energy Agency, 2016)

On peut classer l'énergie en trois groupes principaux selon le type de la source :

Énergie de flux : puissantes et renouvelables mais intermittentes. On peut citer les energies issues du rayonnement solaire, le vent et le mouvement des vagues.

1. La chine et les pays membres de l'OECD ne sont pas inclus dans Asie.
2. Inclut les bunkers maritimes et d'aviation internationaux.

Énergie semi-dense ou intermédiaire : partiellement ou lentement renouvelable mais plus accessible que les énergies de masse. On peut citer la biomasse, l'écoulement de l'eau des rivières, la géothermie et le mouvement des marées.

Énergie dense : Accessible mais non renouvelable et polluante. Issue de l'énergie solaire fossilisée.

Comme on peut voir sur la figure 4.2, la génération d'électricité mondiale dépend principalement sur les ressources non renouvelables, bien que les sources renouvelables soient devenues le premier choix en termes d'expansion et d'évolution.

L'intermittence des ressources renouvelables, le coût d'installation et l'espace requis pour produire une quantité considérable et le manque de choix en lieux propices pour développer certaines ressources renouvelables jouent en leur défaveur. Ainsi, les activités manufacturières continueront à compter sur l'électricité générée par les combustibles fossiles car aucune autre source d'énergie ne peut satisfaire la demande en terme de disponibilité et quantité.

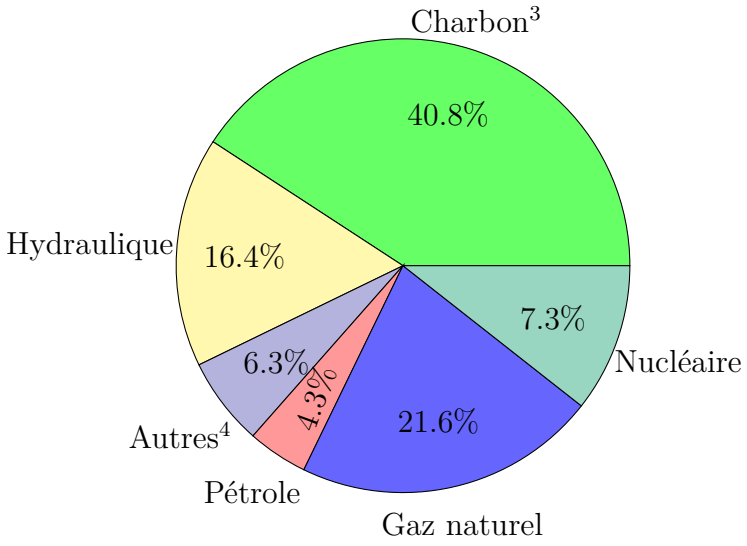


FIGURE 4.2 – Génération d'électricité par ressource énergétique en 2014, (International Energy Agency, 2016)

Il existe quatre secteurs principaux consommant de l'énergie (International Energy Agency, 2016) :

Industriel : qui couvre la fabrication de produits finis, l'exploitation minière, extraction de matières premières et la construction.

Domestique : Chauffage, électronique, électroménager, etc.

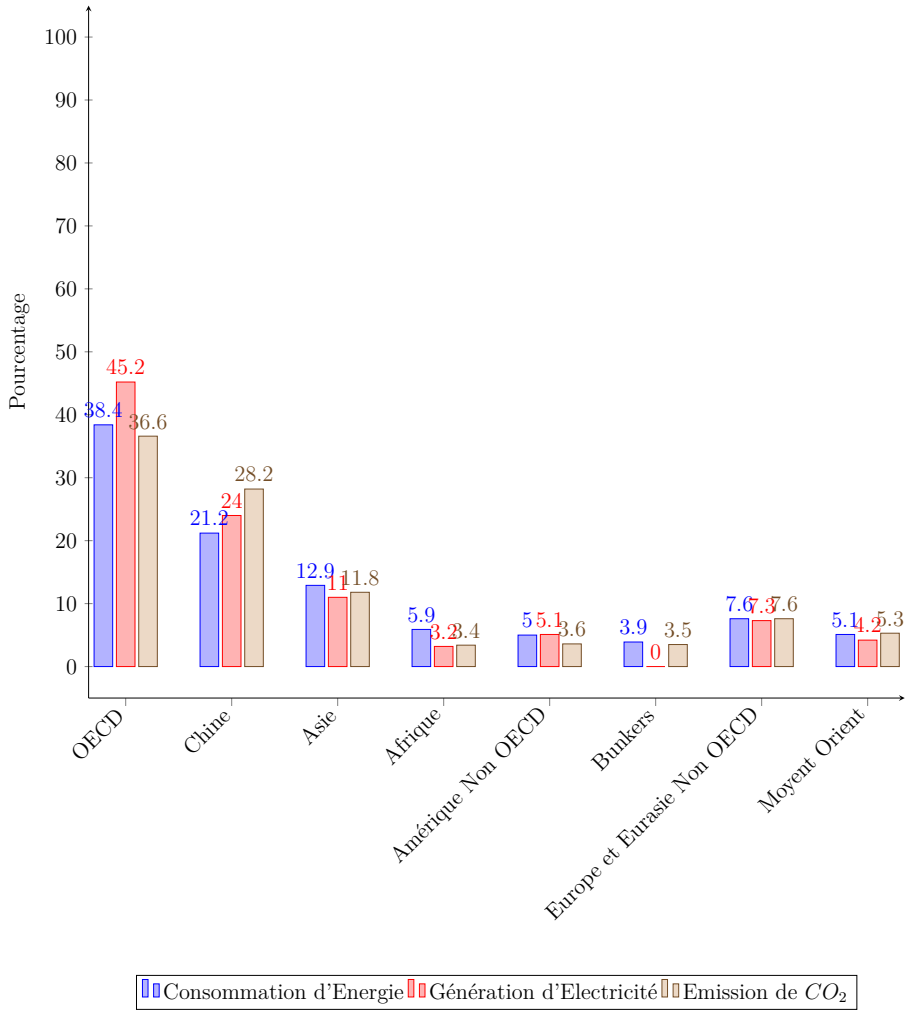
Transports

Services tels que l'éclairage public.

Il est établi que l'énergie provenant de sources non renouvelables contribue à l'émission de gaz à effets de serre responsable du changement climatique.

L'effet de serre est un phénomène normal et naturel mais seule la présence en quantités excessives de certaines substances et leur impact sur les changements climatiques le rendent dangereux.

3. La tourbe et le schiste bitumineux sont inclus.
4. Inclut géothermie, solaire, vent, chaleur, etc.



L'effet de serre est dû à la présence dans l'atmosphère de certains gaz dont les principaux sont : La vapeur d'eau, le gaz carbonique (CO_2), le méthane (CH_4) et le protoxyde d'azote (N_{2O}). Certains gaz sont uniquement produits par l'activité humaine : L'hexafluorure de soufre (S_6F_6) et les perfuorocarbones (PFC_s). La figure 4.1 illustre le lien entre la consommation énergétique, la génération d'électricité et l'émission dans l'air de CO_2. Afin de connaître l'influence d'un gaz à effet de serre sur le réchauffement climatique, on utilise un indice appelé pouvoir de réchauffement global à 100 ans et se mesure par rapport à celui du dioxyde de carbone. Cet indice permet de savoir combien de fois ce gaz contribue à l'effet de serre par rapport à la même quantité de Co_2 100 ans après son rejet dans l'atmosphère.

Avec la croissance de la demande en énergie et les préoccupations écologiques, utiliser l'énergie plus efficacement est devenu primordial.

Nous présentons dans ce qui suit un état de l'art sur la gestion de l'énergie dans le milieu industriel.

4.2 Gestion de l'énergie

Afin de gérer la consommation d'énergie dans le milieu industriel, une connaissance approfondie de la consommation effective lors de toute la phase de production doit être établie. Plusieurs recherches ont été faites dans ce sens, on peut citer Hu et al. (2012), Kant and Sangwan (2015),

W. Li et al. (2011).

Différentes stratégies ont été développées dans l'industrie afin de diminuer la consommation énergetique (Pechmann and Schöler, 2011) :

- . Utilisation de machines plus économiques. Par exemple, une diminution de la taille des composants diminue la consommation en offrant une tension d'alimentation plus faible
- . Coupures d'énergie pour réduire la demande d'énergie maximale, limiter le nombre de changements d'état.
- . Prévisions des charges de l'énergie pour réduire les coûts.

Mais ces méthodes ne sont pas assez effectives car leur impact sur l'énergie totale consommée est minime. Pour y remédier, on opte pour des méthodes et approches pour gérer la consommation energétique par une organisation intelligente de la production.

Plusieurs stratégies se rapportant à la phase de production dans le but de gérer l'énergie ont été mises en place (Pechmann and Schöler, 2011) :

- Considérer la consommation d'énergie comme objectif et la productivité en contrainte ou la productivité en objectif et la consommation d'énergie en objectif, ou considérer l'optimisation bi-objectif consommation d'énergie et productivité.
- Utilisation de modèles statistiques et probabilistes, outils de gestion de risques et analyse de données dans les modèles où le coût énergétique n'est pas simple à prédire en plus d'algorithmes d'optimisation à long terme.
- Modélisation de la consommation d'énergie en variant les vitesses sur différents sous-systèmes.

Plusieurs chercheurs ont proposé des techniques de minimisation d'énergie (DfEM) lors de la fabrication d'un produit depuis la préproduction à la postproduction. On peut citer en exemple Mouzon and Yildirim (2008), Seow et al. (2016).

4.2.1 Une seule machine

Mouzon (2008) a étudié le problème suivant : Soit à ordonnancer n tâches sur une machine. En considérant que chaque tâche est définie par une date de disponibilité r_i et une date limite $\tilde{d}_i$ et une taille p_i. L'objectif est de minimiser la consommation d'énergie et le makespan. Comme $\sum_{i=1}^{n} p_i$ est constante et ne change pas en changeant l'ordonnancement, la minimisation de la consommation d'énergie revient à chercher à minimiser l'énergie consommée au démarrage et aux temps de veille de la machine. Notons ce problème $1|r_i, \tilde{d}_i|C_{max}, TEC$.Mouzon (2008) propose deux algorithmes MOGA et GRASP, ainsi qu'une combinaison des deux algorithmes. Mouzon and Yildirim (2008) ont proposé un modèle basé sur GRASP pour le problème $1|r_i, \tilde{d}_i|TEC, T_{max}$.

Pour le problème $1|r_i, \tilde{d}_i|\epsilon TEC, \bar{T}_w$ où $\bar{T}_w$ représente le retard moyen pondéré et ϵ l'empreinte carbone par unité d'énergie, Liu and Huang (2014) présentent des algorithmes basés sur les métaheuristiques AMGA, NSGA-II.

Augustine et al. (2008) proposent un algorithme en ligne $3 + \sqrt{2}$ compétitif pour le problème $1|power\ down^5|TEC$.

5. On parle d'ordonnancement power-down lorsqu'on peut mettre en veille la machine lorsqu'elle n'est pas active.

Sur un processeur à vitesse variable

Yao et al. (1995) ont considéré le modèle suivant : Soit $[t_0, t_1]$ un intervalle de temps. Une instance du problème d'ordonnancement est un ensemble J de tâches à exécuter sur l'intervalle $[t_0, t_1]$, la préemption est autorisée. A chaque tâche i est associée une date de disponibilité r_i, une date limite $\tilde{d}_i$ et une taille p_i.

Un ordonnancement est la donnée de pairs $S = (s(t), t\hat{a}che(t))$ de fonctions définies sur $[t_0, t_1]$ par

- $s(t)$ est la vitesse du processeur à l'instant t.
- $t\hat{a}che(t)$ donne la tâche exécutée à l'instant t (ou 0 si le processeur est inactif).

Un ordonnancement est réalisable pour l'instance J si l'ordonnancement S vérifie

$$\int_{r_i}^{d_i} s(t)\delta(tache(t), i)\mathrm{d}t = p_i$$

pour tout $i \in J$ où $\delta(x, y) = \begin{cases} 1 & x = y \\ 0 & \text{sinon} \end{cases}$.

La puissance ou l'énergie consommée par unité de temps est une fonction croissante de la vitesse supposée convexe, l'énergie totale consommée étant l'intégrale de la puissance par rapport au temps sur l'intervalle $[t_0, t_1]$.

Yao et al. (1995) proposent un algorithme hors-ligne *YDS* en temps $O(n^3)$ pour le problème $1|r_i, d_i, pmtn|TEC$ dont le principe est le suivant :L'algorithme commence par identifier l'intervalle dit critique où la valeur $g(I) = \frac{\sum_{a \leq r_i, d_i \leq b} p_i}{b-a}$, $I = [a, b]$, est maximale puis ordonnance les tâches dont $[r_i, d_i] \subseteq [a, b]$ selon EDF et à vitesse $\max g(I)$. Un nouveau probème est ensuite construit en supprimant l'intervalle et les tâches déjà étudiés et il détermine le prochain intervalle critique. Bansal and Pruhs (2005) ont prouvé que l'algorithme est optimal. Li et al. (2014) ont proposé un algorithme en temps $O(n^2)$ améliorant le principe de l'algorithme YDS.

Yao et al. (1995) ont aussi proposé deux algorithmes en ligne : *AVR* et *Optimal available*. On donne ci-dessous le principe de chaque algorithme :

AVR : A tout instant t, exécuter le tâche de date limite la plus proche à la vitesse $\sum_i \frac{p_i}{d_i - r_i}$ où la somme est faite sur tous les tâches actifs à cet instant.

Optimal available : A chaque instant t, exécuter le tâche de date limite la plus proche avec la vitesse $\max_x \frac{p(t)}{t}$ où $p(t)$ est la charge inachevée qui a une échéance dans les t prochaines unités de temps.

Bansal et al. (2011) ont prouvé que pour une fonction puissance définie par $p(s) = s^\alpha$ où $\alpha > 1$, l'algorithme AVR est au plus $2^{\alpha-1}\alpha^\alpha$-compétitif et Optimal available est 4-compétitif lorsque $p = 2$. En considérant que la fonction puissance est $P(s) = s^\alpha$ où $\alpha > 1$, Bansal et al. (2012) proposent une classe d'algorithme en-ligne appelée *qOA* et Bansal et al. (2007a) proposent un algorithme hors-ligne et un algorithme en ligne, appelé *BKP* dont le principe est expliquée ci-dessous :

Principe de l'algorithme : A chaque instant t, ordonnancer le tâche inachevé de date limite la plus proche à la vitesse $\max_{t'>t} \frac{\sum_{T_i \in T : r_i \leq t} p_i}{t' - t}$ durant la période $[et - (e-1)t', t']$.

Huang and Ott (2014) proposent un algorithme hors-ligne polynomial optimal basé sur YDS. Atkins et al. (2011) considèrent le problème en ajoutant le facteur température et étudient les algorithmes AVR, Optimal et YDS dans ce cas.

Lorsque la préemption n'est pas autorisée, le problème $1|r_i, d_i, s^\alpha|TEC$ est prouvé NP-difficile et en considérant le cas où l'instance admet des dates échues agréables[6], le problème est dans P (Antoniadis and Huang, 2013).

Antoniadis and Huang (2013) ont montré qu'il existe un PTAS d'approximation $2^{5\alpha-4}$ pour le problème $1|r_i, d_i, s^\alpha|TEC$ et dans le cas où l'instance est laminaire[7], il existe un PTAS d'approximation $2^{4\alpha-3}$. Huang and Ott (2014) ont montré qu'il existe un FPTAS dans le cas où l'instance est purement laminaire[7] et ont présenté un algorithme polynomial dans le cas où les tailles des tâches sont identiques.

On suppose que $\alpha > 1$, Albers and Fujiwara (2007) montrent que, pour le problème $1|r_i, s^\alpha|TEC, F$, il n'existe aucun algorithme en ligne ayant un rapport de compétitivité constant et que le problème hors ligne peut être résolu en temps polynomial. Des algorithmes en ligne ont été proposé par Bansal et al. (2007b) pour le cas où la préemption est autorisée et par Bansal et al. (2008) dans le cas où la vitesse est bornée, ces algorithmes sont améliorés par Lam et al. (2008b). Bansal et al. (2009) proposent un algorithme en ligne dans le cas où la fonction puissance est une fonction convexe quelconque.
Baptiste et al. (2012) ont proposé un algorithme hors-ligne polynomial en temps $O(n^5)$, noté *BCD*, pour le problème $1|r_i, d_i, pmtn, sleep|TEC$ qui est $O(n^4)$ dans le cas des tailles de tâches unitaires. Albers and Antoniadis (2014) ont montré que le problème $1|r_i, d_i, pmtn, power-down|TEC$ est NP-difficile et ont développé un algorithme hors-ligne de facteur d'approximation 4/3 combinant les algorithmes *YDS* et *BCD*, ce facteur est réduit dans le cas où $P(s) = s^\alpha + \lambda$. Pour le cas de fonctions puissance convexes quelconques. Han et al. (2010) et Kumar and Shannigrahi (2015) ont proposé des algorithmes en-ligne notés respectivement *SOA*, *SqOA*.
Bunde (2006) a proposé un algorithme hors-ligne pour le problème $1|r_i, d_i, pmtn, E|C_{max}$ qui est adaptable pour le cas de vitesses discrètes. Les résultats tiennent aussi pour le cas non préemptif et a prouvé qu'il n'existe aucun algorithme exact pour le problème $1|r_i, d_i, E|F$ et présente un algorithme hors-ligne pour le problème $1|p_i = p, r_i, d_i|TEC, F$ étendant celui de Pruhs et al. (2006).
Pour le problème $1|r_i, w_i, pmtn|F_w, TEC$, Antoniadis et al. (2014) proposent un algorithme hors-ligne polynomial dans le cas où la vitesse est discrète et Chan et al. (2010) proposent un algorithme en ligne, noté WLAPS, pour le cas continu. Lorsque la fonction puissance est strictement concave, Andrew et al. (2009) proposent un algorithme en ligne basé sur SRPT. Le problème $1|w_i, E|F$ est prouvé NP-difficile et des algorithmes FPTAS ont été proposés par Megow and Verschae (2012). Chan et al. (2009) ont considéré le problème $1|r_i, d_i|TEC, N_t$, où la vitesse est limitée et proposent un algorithme en ligne, appelé *FSA* dont la complexité a été améliorée par Bansal et al. (2008).
En considérant que la machine varie sa vitesse dans un ensemble de d vitesses discrètes , Li and Yao (2005) proposent un algorithme hors ligne en temps $O(dn \log n)$ pour la minimisation d'énergie et prouvent qu'il est optimal pour un nombre d fixe de vitesses dont la complexité a été améliorée

6. On dit qu'une instance admet des dates échues agréables si pour tout couple de tâches T_i, T_j, on a $r_i \leq r_j \Rightarrow d_i \leq d_j$.

7. On dit qu'un ensemble de tâches est un ensemble laminaire si pour tout couple de tâches T_i, T_j, on a soit $[r_i, d_i) \subseteq [r_j, d_j)$, $[r_j, d_j) \subseteq [r_i, d_i)$ ou $[r_i, d_i) \cap [r_j, d_j) = \emptyset$. Il est purement laminaires si $[r_1, d_1[\subseteq [r_2, d_2[\subseteq \cdots \subseteq [r_n, d_n[$.

par Li et al. (2014).

4.2.2 Machines parallèles

Un modèle mathématique en nombres entiers mixte pour le problème $R_m||TEC, C_{max}$ est proposé par Che et al. (2017) et May et al. (2015) ont proposé un algorithme génétique obtenu en fusionnant et remodelant NSGA-II et SPEA-II pour le problème $R_m|power-down|C_{max}, TEC$. Wang et al. (2018) ont présenté une méthode ϵ-contrainte et une métaheuristique basée sur NSGA-II.
Li et al. (2016), Nicolo et al. (2017) ont considéré le problème $R_m|r_i, d_i|TEC$ et Mouzon (2008) a présenté un algorithme basé sur MOGA pour le problème $R_m|r_i, d_i|C_{max}, TEC$. Chan et al. (2013a) ont présenté un algorithme en-ligne *WPOOL* pour le problème
$R_m|r_i, online, sleep|F_w, TEC$.
Moon et al. (2013) ont proposé une heuristique basée sur l'algorithme génétique pour le problème $R_m|d_i|C_{max} * cost + TEC$ dont le modèle mathématique a été amélioré par Koo and Kim (2016). Cheng et al. (2017), Ding et al. (2016) ont étudié le problème $R_m|C_{max} < B|TEC$.

Processeurs multiples

Bunde (2006) a montré que le problème $P_m|s^\alpha|TEC, F$ est NP-difficile et a proposé un algorithme basé sur *IncMerge* lorsque les tailles des tâches sont identiques. Chan et al. (2011) ont proposé un algorithme en-ligne *POOL* pour le problème $P_m|r_i, pmtn, sleep|TEC$.
Chen et al. (2011) ont prouvé qu'il existe un ordonnancement optimal pour le problème
$Q_m|d_i, E|C_{max}$ dans le cas où la fonction puissance est une fonction convexe quelconque. Pruhs et al. (2006) ont montré qu'il n'est pas possible d'avoir une bonne approximation du makespan optimal pour le problème $Q_m|prec, E|C_{max}$ et ont proposé un algorithme dont le temps d'exécution dépend du nombre de machines. H.L. et al. (2013) ont présenté un algorithme en-ligne, noté *WPOOL*, pour le problème $Q_m|r_i, w_i, power-down|F_w, TEC$.

Albers and Antoniadis (2014) ont prouvé que le problème $Q_m|r_i, d_i, powerdown, pmtn|TEC$ est NP-difficile. Baptiste et al. (2012) ont proposé un algorithme hors-ligne polynomial tandis que Antoniadis et al. (2015) ont proposé un FPTAS. En supposant que $\frac{P(s)}{s}$ est aussi convexe, Irani et al. (2007) ont proposé un algorithme hors ligne basé sur *YDS* et un algorithme en ligne basé sur *AVR*. Bampis et al. (2014) ont présenté un algorithme hors-ligne 2-approché pour le problème $Q_m|r_i, d_i, prec|$.

Albers et al. (2007) ont prouvé que le problème $Q_m|r_i, d_i, prec|$ hors-ligne est NP-difficile où ils ont proposé un algorithme optimal *RR* qui assigne les tâches aux processeurs selon round robin puis sur chaque processeur, l'algorithme de Li and Yao (2005) est appliqué. Les auteurs ont aussi proposé un algorithme en ligne *RR-ON* combinant *RR* avec l'algorithme de Bansal et al. (2007a) dans le cas où les dates de disponibilité sont unitaires et les dates limites "agréables". Lorsque les dates de disponibilité et les dates limites sont arbitraires et que les tailles sont unitaires, Albers et al. (2007) ont prouvé que le problème est NP-difficile au sens fort et ont développé un algorithme polynomial hors-ligne *CRR*. Les auteurs ont proposé un algorithme en ligne *CRR-ON* combinant *RR* avec *AVR*. Albers et al. (2007) ont proposé un algorithme polynomial hors ligne *EDL* dans le

cas général. Pour des fonctions puissances convexes quelconques, Li et al. (2006) ont proposé des algorithmes hors-lignes *OVS*.

Bingham and Greenstreet (2008) ont prouvé que le problème hors-ligne $Q_m|r_i, d_i, pmtn, migration|TEC$ se résolvait polynomialement. Angel et al. (2012) ont proposé un algorithme hors-ligne *BAL* qui se résout en temps $O(nf(n)\log(P))$ où $f(n)$ est la complexité de calcul d'un flot maximum dans un graphe arborescent et P est le rapport entre le nombre de valeurs possibles des vitesses du processeur et la fréquence souhaitée.

Bampis et al. (2013) ont proposé un algorithme hors ligne pour le problème $Q_m|r_i, d_i|TEC$ lorsque les instances des tâches sont agréables[5] dont le temps de complexité dépend du nombre de machine et dans le cas général, il dépend de m et de n. Albers et al. (2016) ont proposé un algorithme en ligne pour des fonctions puissances arbitraires *AVR-hétérogène* pour le problème $R_m|r_i, d_i|TEC$.

Bampis et al. (2014) ont présenté un algorithme hors-ligne pour le problème $Q_m|r_i, d_i, prec|TEC$ où la relation de précédence est représentée par un graphe acyclique.

Lam et al. (2008a) ont proposé un algorithme en ligne $CRR_\gamma - BPS_\epsilon$ pour le problème $Q_m|r_i, d_i, pmtn|TEC$.
Chan et al. (2013b) ont proposé un algorithme *SATA* pour le problème $Q_m|d_i, pmtn, migration, online|F, TEC$.

Gupta et al. (2011) ont présenté un algorithme en-ligne où ils utilisent le principe *HDF* pour l'ordonnancement des tâches dans le problème $Q_m|r_i, d_i, pmtn|F_w, TEC$ et un algorithme en-ligne qui utilise le principe *SRPT* pour l'ordonnancement des tâches dans le problème $Q_m|r_i, d_i, pmtn|F, TEC$.

Certains auteurs ont considérés des problèmes sur des tâches périodiques tels que Cataldo et al. (2015), Piao and Park (2015).

4.2.3 Machines spécialisées

Lorsque l'ensemble des vitesses des machines est discret, Fang et al. (2013) ont donné deux formulations mathématiques du problème $F_m|E|C_{max}$ et ont prouvé qu'il existe toujours un ordonnancement optimal sans délai et proposèrent des heuristiques dans le cas où la capacité de stockage est illimité et montrèrent que le problème peut être transformé en un problème du voyageur de commerce asymétrique lorsqu'il n'y a pas de stockage intermédiaire. Lorsque la vitesse est continue, Fang et al. (2013) ont prouvé que le problème $F_2|no-buffets, E|C_{max}$ est un cas spécial du problème du voyageur de commerce asymétrique.

Liu et al. (2014) ont appliqué l'algorithme d'optimisation multi-objectif NSGA-II pour le problème $F_m|w_i|T_w, TEC$. May et al. (2015) ont proposé un algorithme génétique basé sur NSGA-II et SPEA-II pour le problème $J_m|power-down|C_{max}, TEC$. Dai et al. (2014), Escamilla et al. (2014) proposent un algorithme génétique modifié pour le problème $F_m||C_{max}, TEC$.
Masmoudi et al. (2015) ont présenté deux programmes linéaire et non linéaire en nombre entier mixte pour résoudre le problème $F_m||TEC, cost$.

5. On dit que l'instance est agréable si la propriété suivante est vérifiée : Pour toutes tâches T_i, T_j, si $r_i \leq r_j$ alors $d_i \leq d_j$

4.2.4 Ordonnancement flow shop hybride

Luo et al. (2013) ont considéré le problème $HF(P_{m_1}, P_{m_2}, \ldots, P_{m_L})||TEC, C_{max}$ et ont présenté une métaheuristique basée sur l'algorithme de colonies de fourmis. Pour ce même problème, Dai et al. (2013) ont présenté un algorithme combinant l'algorithme génétique et l'algorithme du recuit simulé. Jiang et al. (2014) ont proposé l'algorithme NSGA-II pour résoudre le problème $HF(P_{m_1}, P_{m_2}, \ldots, P_{m_L})||TEC, C_{max}, cost$. Keller et al. (2015) ont proposé un algorithme de recuit simulé pour résoudre le problème $HF(P_{m_1}, P_{m_2}, \ldots, P_{m_L})||TEC$. Mokhtari and Hasani (2017) ont présenté un modèle mathématique et ont proposé un algorithme combinant le récuit simulé et l'algorithme génétique pour le problème $HF(P_{m_i})_{1 \leqslant i \leqslant L}|maintenance|TEC, C_{max}$
Fang et al. (2016) ont formulé un programme en nombres entiers mixte non linéaire et proposé un algorithme d'essaims particulaires pour $HF(P_{m_1}, P_{m_2}, \ldots, P_{m_L})||vitesse\,de\,coupe, C_{max}$. Liu and Huang (2014) ont implémenté l'algorithme NSGA-II qui identifie l'ensemble des ordonnancements efficaces approchés et un algorithme AMGA pour générer le front pareto qui valide les résultats obtenus pour le problème $HF(1, P_2)|d_i|TEC, T_{max}$. Lam et al. (2008b) ont proposé un programme mathématique mixte pour le problème bi objectif de minimisation du makespan et de l'énergie et proposent une méthode de résolution multi-objective basée sur l'enseignement-apprentissage (teaching-learning).
Stock and Seliger (2015) ont proposé une métaheuristique AISO pour résoudre le problème $HF(P_{m_i})_{1 \leqslant i \leqslant L}|\text{É}nergie\,renouvelable|TEC$.

Chapitre 5

Sur un problème de minimisation d'énergie sur des machines générales et tâches déteriorables

Ce travail a fait objet d'une publication (Tigane et al., 2019).

L'étude considérée est inspirée du processus de laminage à chaud. Il traite du problème de planification de traitement de plaques chaudes dans une ligne de production de laminage à chaud.

5.1 Présentation du problème

Soit un ensemble de m machines parallèles générales et un ensemble de n tâches indépendantes non préemptives disponibles à $t = 0$. L'inactivité des machines n'est pas permise. Le temps d'exécution d'une tâche, noté p_i est une fonction linéaire croissante qui dépend du temps de début d'exécution et est donnée par l'équation suivante :

$$p_{ij} = p_{ij}^0 + \alpha_{ij} t_{ij} \tag{5.1}$$

où $p_{ij}^0 > 0$ est le temps d'exécution basique de la tâche i sur la machine j, $\alpha_{ij} > 0$ est le taux de détérioration et $t_{ij} \geqslant 0$ est le temps de début d'exécution de la tâche i sur la machine j.

On suppose que la détérioration d'une tâche s'arrête dès que son exécution sur la machine commence et que le taux de détérioration est indépendant de la machine, c-à-d. $\alpha_{ij} = \alpha_i, \forall j \in \{1, \ldots, m\}$.

L'objectif est de minimiser le makespan et la consommation énergétique totale.

Le temps de fin d'exécution d'une tâche i sur la machine j est $C_{ij} = t_{ij} + p_{ij}$.

Puisqu'il n'y a aucun temps d'arrêt machine entre l'exécution des tâches, le temps de fin d'exécution sur une machine j est alors $C_j = \sum_{i \in J_j} p_{ij}$, où J_j est l'ensemble des tâches exécutées sur la machine j.

On considère que la machine j consomme δ_j energie par unité de temps durant l'exécution. Ainsi, l'énergie consommée pour l'exécution d'une tâche i est $p_{ij}\delta_j$, et on suppose que cette énergie augmente en fonction du temps avec un taux de θ_j.

L'énergie totale consommée lors de l'exécution d'une tâche i par une machine j est donnée par :

$$E_{ij} = p_{ij}\delta_j + \theta_j t_{ij} \tag{5.2}$$

Par conséquent, la consommation d'énergie totale de la machine j est :

$$E_j = \sum_{i=1}^{n}(p_{ij}\delta_j + \theta_j t_{ij}) \tag{5.3}$$

Le modèle mathématique

En utilisant la notation de Graham et al. (1979), le problème considéré peut être noté par $R_m|p_{ij} = p_{ij}^0 + \alpha_i s|C_{max}, TEC$.

Puisque les conditions sur les machines et les tâches sont les mêmes que dans Mazdeh et al. (2010), on garde les mêmes notations pour les variables de décision et la formulation des contraintes utilisées dans Mazdeh et al. (2010), Mouzon (2008).

Les deux fonctions objectifs considérées dans ce problèmes sont formulées comme suit :

- La minimisation du makespan C_{max} :

$$\min C_{max} = \min \max_{j\in\{1,\dots,m\}}\{C_{max}^{(j)}\} = \min \max_{j\in\{1,\dots,m\}}\{\sum_{i=1}^{n} p_{ij}x_{ij}\}, \tag{5.4}$$

- La minimisation de la consommation d'énergie totale TEC :

$$\min TEC = \min \sum_{j=1}^{m} E_j = \min \sum_{j=1}^{m}(\sum_{i=1}^{n}(p_{ij}\delta_j + \theta_j t_{ij})x_{ij}. \tag{5.5}$$

Avant de présenter les équations représentant les contraintes, on présente les paramètres et variables de décision utilisés.

Les paramètres utilisés représentent ce qui suit :

- n : le nombre de tâches.
- m : le nombre de machines.
- i : l'indice pour la tâche, $i = 1, \dots, n$.
- j : l'indice pour la machine, $j = 1, \dots, m$.
- $i = 0$: l'indice pour une tâche additionnelle.
- t_{0j} : le temps de début d'exécution pour la tâche additionnelle.
- t_{ij} : le temps de début d'exécution d'une tâche i sur la machine j.
- p_{ij} : le temps d'exécution d'une tâche i sur la machine j.
- C_{ij} : le temps de fin d'exécution d'une tâche i sur la machine j.
- E : consommation d'énergie totale.
- C_{max} : temps de fin d'exécution maximale (makespan).

et les variables de décision sont :

$$x_{ij} = \begin{cases} 1 & \text{Si } T_i \text{ est exécutée par machine } M_i, \\ 0 & sinon \end{cases} \tag{5.6}$$

$$y_{ikj} = \begin{cases} 1 & \text{si } T_k \text{ suit immédiatement } T_i \text{ sur machine } M_j, \\ 0 & sinon \end{cases} \tag{5.7}$$

Les contraintes sont formulées comme suit :

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, \ldots, n\} \tag{5.8}$$

$$(t_{ij} + p_{ij})x_{ij} = t_{kj}y_{ikj}, \quad \forall i, k \in \{1, \ldots, n\}, k \neq i, j \in \{1, \ldots, m\} \tag{5.9}$$

$$\sum_{k=1}^{n} y_{0kj} \leq 1, \quad \forall j \in \{1, \ldots, m\} \tag{5.10}$$

$$\sum_{k=1,k\neq i}^{n} y_{ikj} \leq x_{ij}, \quad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \tag{5.11}$$

$$\sum_{i=0,i\neq k}^{n} y_{ikj} = x_{kj}, \quad \forall k \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \tag{5.12}$$

$$\sum_{i=0,i\neq k}^{n} \sum_{j=1}^{m} y_{ikj} = 1, \quad \forall k \in \{1, \ldots, n\} \tag{5.13}$$

$$\sum_{i=1}^{n} p_{ij}x_{ij} \leq C_{max}, \quad \forall j \in \{1, \ldots, m\} \tag{5.14}$$

$$t_{0j} = 0, \quad \forall j \in \{1, \ldots, m\} \tag{5.15}$$

$$t_{ij} \geq 0, C_{ij} \geq t_{ij}, \quad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \tag{5.16}$$

$$x_{ij} \in \{0, 1\}, y_{ikj} \in \{0, 1\} \tag{5.17}$$

La contrainte 5.8 signifie que chaque tâche est exécutée par une et une seule machine . La contrainte 5.9 signifie que le temps de fin d'exécution d'une tâche sur une machine correspond au temps de début d'exécution de la tâche qui la suit immédiatement, pour préciser qu'il n'existe aucun temps d'arrêt machine.

Pour chaque tâche additionnelle 0 allouée à une machine, seulement une seule tâche peut être exécutée immédiatement après (contrainte 5.10). Ceci signifie aussi que le décideur peut choisir de ne pas exécuter de tâches sur une machine si cette décision entraîne un ordonnancement optimal.

Les contraintes 5.11 et 5.12 montrent que les n tâches sont distribuées sur m machines et si une tâches i est immédiatement suivie par une tâche k sur la machine j, alors les deux tâches i et k appartiennent à m (c-à-d. pas de migrations entre machines).

La contrainte 5.13 signifie qu'une seule tâche peut précéder immédiatement une autre tâche (tâche additionnelle incluse) et les tâches sont exécutées par une seule machine (pas de parallélisme de tâches).

La contrainte 5.14 signifie que le temps d'exécution sur une machine et toujours inférieur au makespan.

La contrainte 5.15 signifie que le temps de début d'exécution d'une tâche additionnelle $s_0 = 0$

pour toutes les machines.

La contrainte 5.16 signifie que le temps de début d'exécution est toujours positif et que temps de fin d'exécution est toujours plus grand que le temps de début d'exécution d'une tâche.

5.2 Algorithme pseudo-exact

L'idée de l'ordre de passage établi dans cet algorithme a été inspirée par le résultat obtenu par Browne and Yechiali (1990) pour le problème mono-objectif de minimisation du makespan sur une machine avec tâches déteriorables.

Théorème 5.1. *L'ordonnancement optimal pour le problème* $1|p_i = p_i^0 + \alpha_i t|C_{max}$ *est obtenu en traitant les tâches dans l'ordre croissant des valeurs* p_i^0/α_i *et le makespan est donné par :* $C_{max} = \sum_{i=0}^{n} p_{[i]}^0 \prod_{j=i+1}^{n} (1 + \alpha_{[i]})$, *où* $p_{[i]}^0, \alpha_{[i]}$ *sont, respectivement, le temps d'exécution initial et le taux de déterioration de la* i^{me} *tâche de l'ordonnancement optimal.*

On présente un algorithme pseudo-exact qui donne le front pseudo-Pareto sur toutes les solutions possibles où l'ordre des tâches sur une machine est pris dans l'ordre croissant des p_{ij}^0/α_i.

Étant donné qu'on a aucune obligation d'utiliser toutes les machines, la taille des solutions possibles est m^n.

En effet, le nombre de possibilités de choisir k machines parmi un ensemble de m machines est C_m^k. De plus, on a $\left\{ {n \atop k} \right\}$ [1] façons de répartir les n tâches sur k machines sélectionnés et $k!$ possibilités d'assigner un sous-ensemble d'une partition à une machine spécifique de la combinaison (nombre de permutations de k éléments). D'où la population est donnée par Hsu (2009) :

$$\sum_{k=1}^{m} C_m^k k! \left\{ {n \atop k} \right\} = m^n \tag{5.18}$$

Algorithme 1 détaille le pseudo-code de l'algorithme pseudo-exact.

1. $\left\{ {n \atop k} \right\}$ représente les nombres de Stirling de second ordre qui comptent le nombre de partitions d'un ensemble à n éléments en exactement k sous-ensembles.

Algorithm 1 Algorithme pseudo-exact

procedure ORDONNANCEMENT(n, m)
  $k \leftarrow 1$
  $F \leftarrow S_0$
  while $k <= m$ **do**
    Obtenir toutes les combinaisons sans répétitions de k machines parmi m
    for `Chaque combinaison obtenue` **do**
      `Obtenir toutes les permutations possibles`
      for `Chaque permutation obtenue` **do**
        `Toutes les partitions de l'ensemble des` n `tâches en` k `sous-ensembles, l'allocation des sous-ensembles aux machines suit l'ordre de la permutation et sur chaque machine, les tâches suivent l'ordre croissant de` p^0_{ij}/α_i`. Chaque partition correspond à un ordonnancement` S
        `Calcul du` C_{max} `et` E `correspondants à l'ordonnancement` S
        for `Chaque ordonnancement` $T \in F$ **do**
          if S `domine` T **then**
            $F \leftarrow (F \cup S)/T$
          else if T `domine` S **then**
            $F \leftarrow F$
          else
            $F \leftarrow F \cup S$
          end if
        end for
      end for
    end for
    $k \leftarrow k + 1$
  end whilereturn F
end procedure

5.2.1 Exemple

On considère un problème d'ordonnancement de 15 tâches sur 5 machines.

La table 5.1 détaille les valeurs des temps d'exécution et des taux de détérioration des tâches.

		Tâches														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Temps d'exécution														
machines	1	7	9	1	4	4	5	8	4	5	4	5	5	9	9	3
	2	8	5	4	5	1	6	7	9	4	9	2	1	6	5	5
	3	5	8	4	2	5	1	2	8	2	8	4	2	2	3	4
	4	7	4	2	3	6	5	3	7	5	7	4	5	5	6	2
	5	8	1	7	6	8	4	5	5	8	8	2	6	3	4	5
		Taux de détérioration														
		0.3	0.4	0.2	0.7	0.8	0.4	0.1	0.3	0.9	0.1	0.5	0.4	0.8	0.4	0.5

TABLE 5.1: Temps d'exécution et taux de détérioration des tâches

La consommation énergétique et le taux de détérioration machine sont donnés dans la Table 5.2.

Machines	1	2	3	4	5
Consommation énergétique	4	8	4	3	5
taux de déterioration	0.2	0.6	0.4	0.3	0.1

TABLE 5.2: Consommation énergétique et taux de détérioration machine

La table 5.3 montre le front pseudo-Pareto de l'algorithme pseudo-exact pour l'instance donnée.

solution	Makespan	Consommation énergétique	Ordonnancement
Solution A	10.72	271.2	$3 \to 8 \to 10$ $5 \to 12 \to 11$ $9 \to 6 \to 1$ $15 \to 4 \to 7$ $2 \to 13 \to 14$
Solution B	15.32	263.656	$3 \to 8 \to 10$ $5 \to 12$ $9 \to 6 \to 13 \to 7$ $15 \to 4 \to 1$ $2 \to 11 \to 14$
Solution C	10.04	279.88	$3 \to 8 \to 10$ $5 \to 12 \to 9$ $6 \to 13 \to 1$ $15 \to 4 \to 7$ $2 \to 11 \to 14$
Solution D	19.852	263.356	$3 \to 8 \to 10$ $5 \to 12$ $9 \to 6 \to 13$ $15 \to 4 \to 1 \to 7$ $2 \to 11 \to 14$

TABLE 5.3: Front pseudo-Pareto de la méthode exacte

La figure 5.1 montre l'ordonnancement de la 3^e solution donnée dans la table 5.3 afin d'illustrer la consommation énergétique et le temps d'exécution total par machine.

Dans la figure 5.1, on remarque que le temps d'exécution des tâches et la consommation énergétique ne vont pas de pair. Par exemple, la fin de traitement des tâches sur la machine M_3 est supérieur à celui de M_2 mais elle a consommé moins d'énergie que cette dernière.

On a évalué le temps de réponse de l'algorithme pseudo exact en faisant varier la taille de l'instance. L'algorithme a été codé dans un environnement Java Netbeans environment et les expériences établies dans un ordinateur portable dont les spécificités techniques sont les suivantes 2.20 GHz Intel

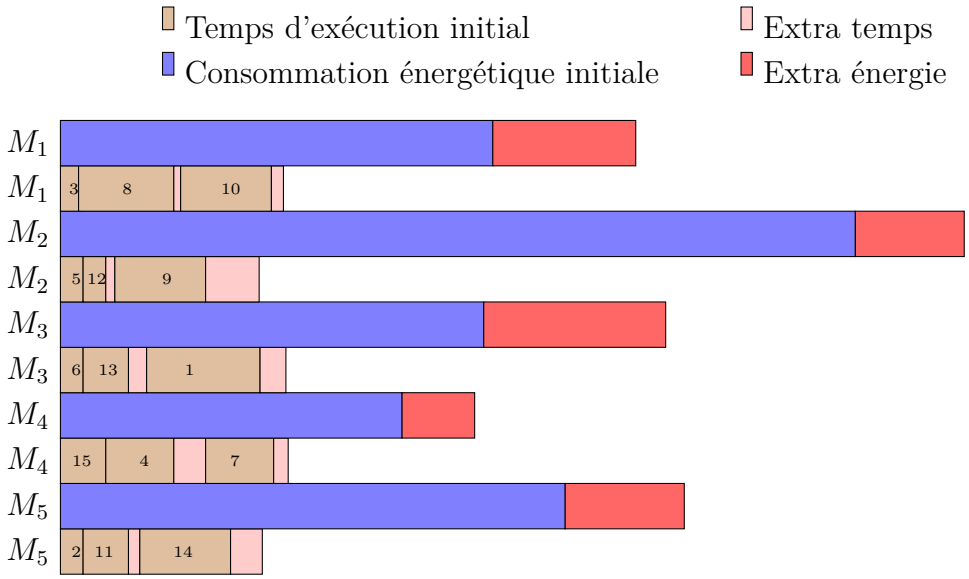


FIGURE 5.1 – Makespan et Consommation énergétique d'une solution

core i5-5200U CPU et 8 GB RAM sous Windows 10.

La table 5.4 illustre les résultats obtenus.

		m		
		3	4	5
n	5	<1s	1s	1s
	8	1 s	1 s	2 s
	10	1 s	4 s	46 s
	12	3 s	57 s	22 min 24 s
	15	7min 51sec	113 min 20 sec	2911 min 59 s

TABLE 5.4: Temps de calcul de l'algorithme pseudo-exact

On observe que le temps nécessaire à l'algorithme pour donner une solution croît rapidement. Ce qui nous mène à considérer une approche métaheuristique.

Si on avait utilisé un algorithme qui prend en considération tout l'ensemble des solutions réalisables, le nombre des solutions augmenterait car pour toute solution réalisable prise en compte par l'algorithme pseudo-exact, on doit considérer les permutations possibles des tâches dans chaque machine. Par exemple, la partition $\{8,1\},\{5,12\},\{9,6,13\},\{15,4,1,7\},\{2,11,14\}$ n'est considérée qu'une seule fois dans l'algorithme pseudo-exact, alors que si on permute dans une machine l'ordre des tâches on obtient une solution réalisable. En considérant cette répartition des tâches sur les machines, le nombre de solutions réalisables qu'on peut obtenir est $(2!)^2(3!)^2 4!$. Ainsi, si on considère toutes les partitions de l'ensemble à 15 éléments en k sous ensembles ($k \in \{1,2,\ldots,5\}$) avec permutations, le nombre total de solutions réalisables augmentera considérablement.

L'expression suivante donne le cardinal de l'espace des solutions réalisables :

$$\sum_{k=1}^{m} \binom{m}{k} k! \left(\sum_{\substack{1 \leqslant n_1 \leqslant n_2 \leqslant \cdots \leqslant n_k \leqslant n \\ n_1+n_2+\cdots+n_k=n}} \binom{n}{n_1, n_2, \ldots, n_k} \frac{n_1! n_2! \ldots n_k!}{\prod_{i=1}^{n-k+1} m(i)} \right) \tag{5.19}$$

où $m(i)$ est égal au nombre de fois où le nombre i apparait dans l'ensemble $\{n_1, n_2, \ldots, n_k\}$, $1 \leqslant k \leqslant m$.

5.3 Approche métaheuristique

5.3.1 OSMS–NSGA-II

Dans cette approche, on implémente la métaheuristique NSGA-II pour résoudre le problème de minimisation du makespan et de la consommation d'énergie totale sur des machines parallèles, où pour chaque individu de la population, l'ordre des tâche sur une machine est fait selon l'ordre croissant de p^0_{ij}/α_i. On appelle cette méthode OSMS–NSGA-II (pour Optimal single machine scheduling NSGA-II).

– *Codage des solutions :* Dans l'algorithme développé, les solutions (chromosomes) sont représentées par une matrice binaire $(n \times m)$, notée $A = (a_{ij})$ où un gène $a_{ij} = 1$ si la tâche i est exécutée par la machine j, et $a_{ij} = 0$ sinon.

La table 5.5 illustre un exemple d'une solution codée $(n = 7, m = 4)$. Selon cet ordonnancement, la tâche 1 est exécutée sur la machine 1 puisque $a_{11} = 1$.

		Machines			
		1	2	3	4
Tâches	1	1	0	0	0
	2	0	1	0	0
	3	0	0	0	1
	4	0	1	0	0
	5	0	0	1	0
	6	1	0	0	0
	7	0	0	1	0

TABLE 5.5: Une solution (ordonnancement) codée

Chaque génération est composée de l'ensemble des parents et des enfants.

– *Opérateurs génétiques :* Chaque ensemble d'enfants C est le résultats des opérateurs de croisement et mutation et de l'aléatoire.

La population des parents choisis pour le croisement et la mutation est choisie grâce à la technique tournoi de surpeuplement dont le principe est repris ci-dessous :

Dans un tournoi de surpeuplement, on doit calculer la distance de surpeuplement, noté par $dist_i$. Le calcul se fait comme suit : Soit $f_k(i)$ la valeur du k^e objectif de l'individu i et r le rang auquel il appartient. On définit :

- f_k^{max} La valeur maximum du k^e objectif sur toute la population ;
- f_k^{min} la valeur minimale du k^e objectif sur toute la population ;
- $f_{k^+}(i)$ la plus proche valeur de $f_k(i)$ dans le r^e groupe non-dominé, où $f_{k^+}(i) \geq f_k(i)$. Si

$f_k(i)$ la valeur maximum du groupe, on pose $f_{k^+}(i) = +\infty$.
S'il existe plus qu'un individu où $f^k(i)$ est de valeur maximale, on pose l'une d'elles $f_{k^+}(i) = +\infty$ et $f_{k^+}(i) = 0$ pour les autres ;

- $f_{k^-}(i)$ est la plus proche valeur de $f_k(i)$ dans le r^e groupe non-dominé, où $f_{k^-}(i) \leqslant f_k(i)$.
  Si $f_k(i)$ est la valeur minimum du groupe, on fixe $f_{k^-}(i) = -\infty$.
  S'il existe plus d'un individu où $f^k(i)$ est de valeur minimale, on pose l'une d'elles $f_{k^-}(i) = -\infty$ et $f_{k^+}(i) = 0$ pour les autres.

La distance de surpeuplement est alors :

$$dist_i = \sum_{k=1}^{2} \frac{f_{k^+}(i) - f_{k^-}(i)}{f_k^{max} - f_k^{min}} \tag{5.20}$$

Soient i, j , on définit la relation d'ordre, notée $i \prec_n j$, comme suit :

$$i \prec_n j \Leftrightarrow r_i < r_j \text{ ou } (r_i = r_j \text{ et } dist_i \geqslant dist_j) \tag{5.21}$$

Ainsi, la nouvelle population P est obtenue en prenant les l individus du classement obtenu grâce à la technique de tournoi de surpeuplement.

Les parents utilisés pour le croisement sont aussi choisis selon ce même classement.

La méthode de croisement nous permet de créer deux enfants à partir de deux parents différents comme suit : Un nombre aléatoire p est choisi dans l'ensemble $\{2, 3, \ldots, n-1\}$, ensuite les $p-1$ premières colonnes du premier parent (respectivement second) parent formera les $p-1$ premières colonnes du premier (respectivement second) enfant et les $n-p-1$ dernières colonnes du premier (respectivement second) parent forment les $n-p-1$ dernières colonnes du second (respectivement premier) enfant.

Les individus à muter sont aussi choisis selon le classement du tournoi de densité et la mutation se fait comme suit : pour chaque individu à muter, on choisit aléatoirement une tâche et on change la machine pour laquelle la tâche était attribuée.

Ci-dessous le pseudo-code de la métaheuristique.

Algorithm 2 Pseudo code OSMS–NSGA-II

Générer une population initiale aléatoire P_0 de taille l
$k \leftarrow 1$
Ordonner les tâches i sur la machine j selon l'ordre croissant de p_{ij}^0/α_i
Évaluer la fonction fitness de chaque solution de la population P_0
Trier les solutions selon la non domination et la distance de voisinage pour obtenir les groupes des solutions non-dominées
while Critère d'arrêt non satisfait **do**
    Créer le groupe des enfants C de taille l par croisement et mutation
    Ordonner les tâches i sur la machine j selon l'ordre croissant de p_{ij}^0/α_i
    Évaluer la fonction fitness de chaque solution de la population $P \cup C$
    Trier les solutions selon la non domination et la distance de voisinage pour obtenir les groupes des solutions non-dominées
    Ranger les solutions dans la population P_k
    $k \leftarrow k + 1$
end while
return Le groupe non-dominé de rang 1

5.3.2 Exemple

La table 5.6 présente un front pseudo-Pareto obtenu par l'algorithme OSMS NSGA-II.

Makespan	Consommation énergétique	Ordonnancement	
Solution I	10.72	271.2	$3 \to 8 \to 10$ $5 \to 12 \to 11$ $9 \to 6 \to 1$ $15 \to 4 \to 7$ $2 \to 13 \to 14$
Solution II	19.852	263.356	$3 \to 8 \to 10$ $5 \to 12$ $9 \to 6 \to 13$ $15 \to 4 \to 1 \to 7$ $2 \to 11 \to 14$
Solution III	15.816	267.344	$3 \to 8 \to 10$ $5 \to 12 \to 11$ $9 \to 6 \to 14 \to 1$ $15 \to 4 \to 7$ $2 \to 13$

TABLE 5.6: Front pseudo-Pareto OSMS NSGA-II

5.4 IS-NSGA-II

– *Codage des solutions :* De légères modifications ont été faites dans la structure du chromosome et de la technique de croisement.

Dans cet algorithme, le chromosome est une matrice $(n \times 2)$, notée (b_{ij}), où b_{i1} représente l'indice de la tâche et b_{i2} représente l'indice de la machine dans laquelle la tâche doit être exécutée.

L'ordre d'exécution dans une machine suit l'ordre d'apparence des tâches dans la première ligne du chromosome.

Machines	2	1	2	3	1
Tâches	4	2	3	1	5

TABLE 5.7: Une solution codée

La table 5.7 illustre un exemple d'une solution codée (5×2) avec 3 machines et 5 tâches.

Par exemple, les tâches 2 et 5 sont exécutées dans cette ordre sur la machine 1.

– *Opérateurs génétiques :* Concernant l'opérateur de croisement, pour chaque couple de parents choisi, on crée deux enfants comme suit : la première ligne du premier(respectivement second) parent forme la première ligne du premier (respectivement second) enfant, ensuite, un nombre aléatoire entre 2 et $(n-1)$, p est choisi, les $p-1$ premières cases de la seconde ligne du premier (respectivement second) parent sont attribuées au premières $p-1$ cases du premier (respectivement second) enfant.

On attribue les cases restantes de la seconde ligne du premier (respectivement second) parent à celles du second (respectivement premier) enfant.

Cette étape nous permet d'éviter l'apparence d'une tâche plus d'une fois.

La technique de mutation utilisée dans cette seconde approche est celle utilisée dans la première approche (OSMS-NSGA-II).

Algorithm 3 Pseudo code IS–NSGA-II

```
Générer une solution initiale aléatoire P_0 de taille l
k ← 1
Evaluer la fonction fitness de chaque solution de la population
Trier les solutions par la non domination et la distance de voisinage pour
obtenir les groupes non-dominées
while Critère d'arrêt non satisfait do
    Créer le groupe des enfants C de taille l par croisement et mutation
    Evaluer la fonction fitness de chaque solution de la population P ∪ C
    Trier les solutions par la non domination et la distance de voisinage
pour obtenir les groupes non-dominées
    Trier les solutions dans la population P_t
    k ← k + 1
end while
return Le groupe non-dominé de rang 1
```

5.4.1 Exemple

L'algorithme IS-NSGA-II a été testé sur la même instance (15-5).

La table 5.8 présente l'exemple d'un front pseudo-Pareto pour cet algorithme.

	Makespan	Consommation énergétique	Ordonnancement
Solution 1	10.72	269.92	$3 \to 8 \to 10$ $5 \to 12 \to 11$ $6 \to 9 \to 1$ $15 \to 4 \to 7$ $2 \to 13 \to 14$
Solution 2	15.32	260.043	$3 \to 8 \to 10$ $5 \to 12$ $9 \to 6 \to 7 \to 14$ $15 \to 4 \to 1$ $2 \to 11 \to 13$
Solution 3	20.052	258.196	$3 \to 8 \to 10$ $12 \to 5$ $6 \to 9 \to 14$ $4 \to 15 \to 7 \to 1$ $2 \to 11 \to 13$
Solution 4	10.04	279.88	$3 \to 8 \to 10$ $5 \to 12 \to 9$ $6 \to 13 \to 1$ $15 \to 4 \to 7$ $2 \to 11 \to 14$
Solution 5	14.96	265.688	$3 \to 8 \to 10$ $5 \to 12 \to 11$ $6 \to 9 \to 7 \to 1$ $15 \to 4 \to 14$ $2 \to 13$

TABLE 5.8: Front pseudo-Pareto IS NSGA-II

On peut observer à partir des front pareto obtenus pour les deux alorithmes que les solutions 1 et 2 de la table 5.8 ont le même makespan que les solutions A et B de la table 5.6 mais la consommation énergétique est meilleure.

On observe que la solution 3 a un makespan pire que toutes les solutions de la table 5.6 mais sa consommation énergétique est meilleure.

On remarque aussi que certaines tâches sont toujours assignées aux mêmes machines ou le temps d'exécution est le plus petit et dont la position selon l'ordre d'ordonnancement choisi est généralement la première.

5.5 TOPSIS

Pour pouvoir déterminer un ordre des solutions de l'ensemble pseudo-Pareto sur un critère de priorité sur les objectifs, on peut utiliser la méthode TOPSIS (pour Order of Preference by Similarity to Ideal Solution). TOPSIS est une méthode multi-critères développée parYoon and Hwang (1981).

Le principe de cette méthode est comme suit : On suppose qu'on a l solutions à k critères (appelés alternatives). La méthode donne, pour chaque alternative, un coefficient entre 0 et 1 basé sur la distance euclidienne entre l'alternative et les alternatives appelées idéales (meilleure et pire).

D'abord, on crée une matrice $(l \times k)$, $X = (x_{ij})$, où x_{ij} est la valeur du j^{eme} critère de la i^{eme} solution.

Ensuite, on normalise la matrice avec la méthode de normalisation $r_{ij} = \frac{x_{ij}}{\sum\limits_{h=1}^{l} x_{hj}^2}$.

On assigne à chaque critère un poids $w_j \in [0, 1]$ tel que $\sum\limits_{j=1}^{k} w_j = 1$, puis on calcule la matrice pondérée normalisée $T = (t_{ij})$ où $t_{ij} = r_{ij} * w_j$.

Pour déterminer la meilleure et la pire alternative, on divise l'ensemble des critères en deux sous-ensembles F_-, F_+.

F_- (respectivement F_+) est l'ensemble des critères ayant un impact négatif (respectivement positif).

Pour chaque critère j de F_- (respectivement F_+), la meilleure alternative est $t_{bj} = \min\limits_{i \in \{1,\dots,l\}} \{d_{ij}\}$ (respectivement $t_{bj} = \max\limits_{i \in \{1,\dots,l\}} \{d_{ij}\}$), la pire alternative est $t_{wj} = \max\limits_{i \in \{1,\dots,l\}} \{d_{ij}\}$ (respectivement $t_{wj} = \min\limits_{i \in \{1,\dots,l\}} \{d_{ij}\}$).

Puis, on calcule les distances d_{ij} entre chaque alternative $i \in \{1, \dots, k\}$ et la meilleure et pire alternatives pour chaque critère j en utilisant la distance de norme L_2. $d_{ib} = \sqrt{\sum\limits_{j=1}^{k} (t_{ij} - t_{bj})^2}$, $d_{iw} = \sqrt{\sum\limits_{j=1}^{k} (t_{ij} - t_{wj})^2}$.

Enfin, on calcule la similarité à la pire alternative s_{iw} : $s_{iw} = \frac{d_{iw}}{d_{iw}+d_{ib}}$, $\forall i \in \{1, \dots, k\}$ et on ordonne les solutions dans l'ordre décroissant de s_{iw}.

5.5.1 Exemple

On applique la méthode Topsis au front pseudo-Pareto obtenu dans la table 5.8 utilisant plusieurs poids.

Le premier poids correspond au critère du makespan et le second à la consommation énergétique.

Poids	0.7-0.3	0.6-0.4	0.5-0.5	0.4-0.6	0.3-0.7
Ordre obtenu	4-1-5-2-3	4-1-5-2-3	1-4-5-2-3	1-4-5-2-3	1-4-5-2-3

TABLE 5.9: Résultats Topsis

5.6 Approche métaheuristique hybride

Les résultats obtenus dans les calculs expérimentaux effectués sur les algorithmes OSMS et IS NSGA-II nous ont menés à considérer une autre approche pour la résolution du problème. Dans cette approche, on propose un algorithme NSGA-II où la population initiale est composée d'individus générés aléatoirement et d'individus créés grâce à des règles d'ordonnancement et d'individus qui sont des mutations des individus créés. Les règles d'ordonnancement sont :

Première règle : Sur chaque machine, les tâches sont classés selon l'ordre croissant de p_{ij}^0/α_i ensuite, à chaque instant t, on choisit la première machine disponible et la première tâche disponible dans le classement obtenue.

Seconde règle : Chaque tâche est assignée à la machine dans laquelle le temps d'exécution basique est le plus petit. Ensuite, dans chaque machine, les tâches sont triées dans l'ordre croissant des p_{ij}^0/α_i.

Troisième règle : Dans chaque machine, les tâches sont triées selon l'ordre croissant des p_{ij}^0/α_i. Ensuite, à chaque instant t, on choisit la plus rapide machine disponible et la première tâche disponible dans le classement.

5.7 Discussion

Les calculs expérimentaux ont été faits sur de petites, moyennes et grandes instances dans les trois approches dont le but est de comparer leur efficacité.

On observe que pour des instances de petites tailles, la différence entre les résultats pour les approches utilisées est minime mais lorsque la taille de l'instance augmente, l'approche hybride donne de meilleurs résultats.

En effet, l'approche hybride a obtenu de meilleures solutions par rapport aux autres solutions dans 34 instances parmi 54 (cellules vertes dans la table 5.10). De plus, certaines des solutions obtenues sont non-dominées par les solutions des autres approches dans 20 instances (cellules jaunes dans la table 5.10).

On remarque aussi que les individus obtenus dans le front pseudo-Pareto sont voisines de l'individu créé suivant la seconde règle.

Poids TOPSIS (makespan - Consommation énergétique)	Instances	Makespan et Consommation énergétique de la meilleure solution		
		OSMS-NSGA-II	IS-NSGA-II	Approche hybride
0.3-0.7	3×10	21.68-29.468	21.68-30.068	21.68-29.468
	3×20	66.364-177.4	88.306-206.543	67.918-168.54
	5×20	22.16-70.7	27.164-69.979	19.751-60.698
	5×30	45.506-150.21	46.728-158.037	36.838-108.428
	8×20	8-28.394	7.68-28.246	7.42-29.358
	8×30	14.696-68.206	17.45-61.442	12.504-57.306
	10×40	20.209-110.797	12.952-77.863	12.952-77.863
	10×50	35.804-176.515	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	8.001-47.561	5.64-45.604
	15×50	9.132-72.646	10.54-74.728	9.152-65.83

	20×50	6.2-49.925	8-55.77	4.99-53.695
	20×60	8.08-79.851	8.3-82.482	7.356-71.54
	15×120	57.768-714.32	64.07-740.675	37.844-608.225
	15×150	119.435-1501.755	129.803-1621.277	61.57-1218.283
	20×150	55.57-803.388	60.748-920.236	27.801-627.554
	20×180	95.12-1621.69	114.977-1764.36	48.752-1175.453
	25×180	51.335-852.61	51.495-986.338	27.425-700.327
	25×200	71.458-1219.765	82.346-1479.756	38.623-1013.234
0.5-0.5	3×10	20.16-32.204	21.68-30.068	18.64-32.8
	3×20	66.364-177.4	84.616-208.56	63.84-172.977
	5×20	22.16-70.7	28.192-74.472	19.751-60.698
	5×30	45.506-150.21	42.254-167.155	36.838-108.428
	8×20	7.6-29.734	7.68-28.246	7.42-29.358
	8×30	14.696-58.206	16.02-64.831	12.334-57.629
	10×40	20.209-110.797	12.952-77.863	12.952-77.863
	10×50	33.473-184.736	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	6.96-51.117	5.64-45.722
	15×50	9.132-72.646	10.54-74.728	7.816-73.48
	20×50	6.2-49.925	6.77-62.162	4.99-53.695
	20×60	7.6-81.996	8.3-82.482	7.356-71.54
	15×120	56.705-722.112	64.07-740.675	37.083-615.285
	15×150	119.435-1501.755	124.146-1655.302	61.57-1218.283
	20×150	51.99-839.691	60.748-920.236	27.801-627.554
	20×180	91.347-1686.64	114.977-1764.36	47.431-1205.122
	25×180	48.864-863.3	50.196-988.87	26.927-710.223
	25×200	70.822-1318.538	78.803-1555.967	38.178-1017.288
0.7-0.3	3×10	20.16-32.204	20.64-33.924	18.64-32.8
	3×20	66.364-177.4	84.616-208.56	63.84-172.977
	5×20	22.16-70.7	24.53-74.521	19.751-60.698
	5×30	41.514-170.897	42.254-167.155	36.838-108.428
	8×20	7.6-29.734	7.68-28.246	7.18-30.846
	8×30	14.696-58.206	16.02-64.831	12.334-57.629
	10×40	20.209-110.797	12.952-77.863	12.952-77.863
	10×50	32.622-190.133	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	6.96-51.117	5.64-45.722
	15×50	9.132-72.646	9.89-78.002	7.816-73.48
	20×50	6.2-49.925	6.77-62.162	4.99-53.695
	20×60	7.6-81.996	8.02-85.537	7.12-73.86
	15×120	56.705-722.122	62.764-773.6	36.427-629.361
	15×150	113.57-1685.15	121.9-1669.534	61.57-1218.283
	20×150	50.234-887.036	58.837-960.626	27.801-627.554
	20×180	90.19-1731.75	106.153-1984.218	46.535-1205.47
	25×180	47.18-920.152	50.196-988.87	26.039-722.354
	25×200	70.265-1324.357	78.803-1555.967	35.772-1095.143

TABLE 5.10: Performances de l'approche hybride

5.8 Heuristique

En se basant sur les observations obtenus lors des comparaisons entre métaheuristiques, on propose l'heuristique suivante :

Algorithm 4 Heuristique

procedure ORDONNANCEMENT(n, m)
    Créer un ordonnancement F suivant la deuxième règle utilisée pour la création des individus pour l'algorithme $IS - NSGA - II$.
    calculer E et C_{max}.
    k l'indice de la machine ayant un temps d'exécution égal à C_{max}.
    D l'ensemble des couples (i_t, j_t) où i_t est l'indice de la tâche exécutée en dernier sur la machine M_{j_t}, $j_t \neq k$, $t \in \{1, \ldots, m-1\}$ représente la position du couple dans l'ensemble D, $|D| = m - 1$.
    $t \leftarrow 0$
    for t from 0 to $m - 1$ **do**
        if Trouver une machine T_l, $l \neq k$ qui en allouant la tâche T_{i_t} à M_l et en réarrangeant les tâches T_i exécutées sur la machine M_l selon l'ordre p_{il}^0/α_i, on obtient $C_l \leq C_{max}$ et $E_l \leqslant E_{j_t}$ **then**
            Attribuer la tâche T_{i_t} à la machine T_l et mettre à jour $E_{j_t}, E_l, C_{j_t}, C_l; E$
        else
            $t \leftarrow t + 1$
        end if
    end forreturn F
end procedure

Chapitre 6

Problème de minimisation d'énergie dans un atelier flow shop hybride avec tâches déteriorables

Soit un atelier flowshop hybride à s étages. Chaque étage $k \in \{1, \ldots, s\}$ est composée de m_k machines parallèles. Soit un ensemble de n tâches, chaque tâche i est constituée de s opérations qui passent par les étages dans le même ordre de passage. Ainsi, La tâche i est traitée à l'étage 1, puis à l'étage 2, et ainsi de suite jusqu'à l'étage s. Chaque machine du même étage peut exécuter au plus une opération à la fois et chaque opération est exécutée par une seule machine. Les temps d'exécution sont déterministes sur certains étages et variables dans d'autres et dépendent de la tâche. Le temps de traitement est formulé sous forme de fonction qui dépend du temps de la façon suivante : Soit t_{ik} le temps de début d'exécution de l'opération O_i à l'étage k, c_{ik} le temps de fin d'exécution de l'opération O_i sur l'étage k (la valeur donnée pour f_{i0} est 0 par convention), et soit D_{ik} l'instant dans lequel la détérioration de la tâche i commence à l'étage k, le temps d'exécution est donné par :

$$p_{ijk} = \begin{cases} p^0_{ijk} & si\ t_{ik} \leq D_{ik} + c_{i(k-1)} \\ p^0_{ijk} + \alpha_{ik}(t_{ik} - D_{ik} - c_{i(k-1)}) & si\ t_{ik} > D_{ik} + c_{i(k-1)} \end{cases} \tag{6.1}$$

où p^0_{ijk} est le temps d'exécution basique de l'opération O_{ik} sur la machine j et α_{ik} est le taux de détérioration. D_{ik} est infini lorsque le temps d'exécution est déterministe. On suppose aussi que le nombre de machines actives d'un même étage simultanément est limité sur certains étages voir tous.

Soit C_{max} le temps de fin d'exécution maximal sur toutes les machines (makespan). e_{jk} l'énergie consommée par unité de temps par la machine en activité j à l'étage k. On suppose que l'énergie consommée pendant que la machine est inactive est négligeable. Soit E_{jk} l'énergie totale consommée par la machine et TEC l'énergie totale consommée alors le calcul de C_{max} et TEC se fait comme

suit :

$$C_{max} = \max_{i\in\{1,\dots,n\}} C_{is} \tag{6.2}$$

$$TEC = \sum_{k=1}^{s}\sum_{j=1}^{m_k} E_{jk} \tag{6.3}$$

6.1 Le modèle mathématique

Le modèle mathématique peut être exprimé de la manière suivante

$$\sum_{j=1}^{m_k} x_{ijk} = 1 \quad \forall i \in \{1,\dots,n\}, k \in \{1,\dots,s\} \tag{6.4}$$

$$\sum_{i=1,i\neq l}^{n} z_{il}^{k} \leq v_k \quad \forall k \in \{1,\dots,s\},\ l \in \{1,\dots,n\} \tag{6.5}$$

$$C_{i1} \geqslant \sum_{j=1}^{m_k} x_{ij1}p_{ij1} \quad \forall i \in \{1,\dots,n\}, k \in \{1,\dots,s\} \tag{6.6}$$

$$C_{ik} \geqslant c_{i(k-1)} + \sum_{j=1}^{m_k} x_{ijk}p_{ijk} \quad \forall i \in \{1,\dots,n\}, k \in \{2,\dots,s\} \tag{6.7}$$

$$c_{lk} - c_{ik} \geqslant c_{ik}\sum_{j=1}^{m_k} p_{ijk}y_{ilj}^{k} + \left((\sum_{j=1}^{m_k} y_{ilj}^{k}) - 1\right).B \quad \forall i,l \in \{1,\dots,n\}, k \in \{1,\dots,s\} \tag{6.8}$$

$$\sum_{i=0,l\neq i}^{n} y_{ilj}^{k} = x_{ljk} \quad \forall l \in \{1,\dots,n\}, j \in \{1,\dots,m_k\}, k \in \{1,\dots,s\} \tag{6.9}$$

$$\sum_{l=1,i\neq l}^{n+1} y_{ilj}^{k} = x_{ijk} \quad \forall i \in \{1,\dots,n\}, k \in \{1,\dots,s\}, j \in \{1,\dots,m_k\} \tag{6.10}$$

$$\sum_{i=0,i\neq l}^{n}\sum_{j=1}^{m_k} y_{ilj}^{k} = 1 \quad \forall j \in \{1,\dots,n\}, k \in \{1,\dots,s\} \tag{6.11}$$

$$z_{il}^{k} = 1 - \min\left[\min\left(\left|E\left(\frac{c_{lk}-c_{ik}}{\sum_{j=1}^{n} x_{ljk}}\right)\right|, \left|E\left(\frac{c_{lk}-c_{ik}}{\sum_{j=1}^{n} x_{ljk}}\right)\right|\right), 1\right] \tag{6.12}$$

$$C_{max} \geq C_{is} \quad \forall i \in \{0,\dots,n\} \tag{6.13}$$

$$x_{ijk}, y_{ilj}^{k}, z_{il}^{k} \in \{0,1\}, \tag{6.14}$$

L'objectif est de minimiser le makespan C_{max} et la consommation énergétique TEC définies par les équations 6.15, 6.16

$$\min C_{max} = \min \max_{i\in\{1,\dots,n\}} C_{is} \tag{6.15}$$

$$\min TEC = \min \sum_{k=1}^{s}\sum_{j=1}^{m_k} E_{jk} = \sum_{k=1}^{s}\sum_{j=1}^{m_k}\sum_{i=1}^{n} p_{ijk}e_{jk}x_{ijk} \tag{6.16}$$

où les paramètres sont

- i, l indices pour les tâches, $i, l \in \{1,\dots,n\}$,
- k indice de l'étage, $k \in \{1,\dots,s\}$,
- m_k nombre de machines à l'étage k,

- j, r indices des machines pour l'étage k, $j, r \in \{1, \ldots, m_k\}$,
- v_k nombre maximal de machines actives simultanément à l'étage k.

Les trois variables de décision utilisées expriment

- x_{ijk} est une variable binaire qui donne 1 si la tâche i est exécutée par la machine j à l'étage k, 0 sinon,
- y^k_{ilj} est une variable binaire qui donne 1 si la tâche i précède immédiatement la tâche l sur la machine j à l'étage k, 0 sinon,
- z^k_{il} est une variable binaire qui donne 1 si $[t_{ik}, c_{ik}] \cap [t_{lk}, c_{lk}] \neq \emptyset$, 0 sinon,

Les contraintes 6.4,6.11 signifient que toute tâche est exécutée sur une seule machine à tout instant (pas de parallélisme, non migration). Contrainte 6.5 exprime la condition sur le nombre maximum de machines actives simultanément à chaque instant. Contrainte 6.6 précise que chaque tâche est entièrement exécutée une fois commencée (pas de préemption) et 6.7 précise qu'une opération d'une tâche doit être complétée au niveau de son étage avant que l'opération de l'étage suivant soit commencée et 6.8 signifie que le début d'exécution d'une tâche est au moins égal à la fin du temps d'exécution de la tâche qui la précède sur la même machine. Contrainte 6.9 signifie qu'au plus une tâche peut suivre immédiatement une tâche fantôme (une tâche fantôme est une tâche fictive qui nous permet d'accepter qu'une machine ne soit jamais utilisée). 6.10 montre que toutes les tâches sont assignées aux machines et que si une tâche est suivie immédiatement par une autre sur une machine alors les deux tâches appartiennent à cette machine. Contrainte 6.13 signifie que la date de fin d'exécution des tâches sur le dernier étage est au plus égale au makespan. L'expression 6.12 nous permet d'obtenir la valeur exacte des z^k_{il}.

6.2 Approches métaheuristiques

6.2.1 NSGA-II

On reprend la métaheuristique NSGA-II utilisée dans le chapitre précédent. Certaines modifications ont été apportées par rapport au modèle.

Codage des solutions : Chaque solution est représentée par une matrice (A_{ij}) de taille $n \times \sum_{i=1}^{s} m_i$, où sur chaque colonne j sont notées les tâches qui sont alloués à la machine j et l'ordre dans lequel ils se trouvent dans la colonne représente leur ordre de traitement. On ajoute aussi un tableau représentant l'ordre de priorité des machines pour chaque niveau.

Création de la population : La population initiale est un mélange d'individus générés aléatoirement et d'individus crées suivant certaines règles.

- Individu généré aléatoirement : L'aléatoire s'opère au niveau 1 où l'allocation des tâches et des machines se fait aléatoirement. L'ordre de priorité des machines est déterminé aléatoirement pour tous les niveaux. A partir du niveau 2, on utilise la règle FIFO en respectant l'ordre des machine préétabli.
- Individu créé : En ce qui concerne les individus créés, l'ordre de priorité des machines est fait selon l'ordre croissant du coût énergétique. En ce qui concerne la solution, cela se fait de 3 manières. Premier individu : on commence par le niveau 1 en attribuant les tâches à la machine

où leur temps d'exécution est le plus petit et sur chaque machine, les tâches sont triés selon l'ordre croissant de leur temps d'exécution. A partir du niveau 2, on utilise la règle FIFO en respectant l'ordre des machine préétabli. Deuxième individu, sur chaque machine du niveau 1, on classe les tâches selon l'ordre croissant de leur temps d'exécution. Ensuite, à chaque instant, on choisit la première machine disponible et la première tâche disponible selon le classement. Troisième individu : sur chaque machine du niveau 1, on classe les tâches selon l'ordre croissant de leur temps d'exécution. Ensuite, à chaque instant, on choisit la première machine ayant consommé le moins d'énergie et la première tâche disponible selon le classement. A partir du niveau 2, on utilise la règle FIFO en respectant l'ordre des machines préétabli.

En ce qui concerne la génération de la population d'enfants, on a chaque population d'enfants C est constituée d'individus crées par croisement ou mutation et d'individus aléatoires. Le choix des parents à croiser ou des individus à muter se fait de façon identique que pour les algorithmes NSGA-II du chapitre précédent. En ce qui concerne les opérateurs génétiques, on a :

- Le croisement : Le croisement se fait au niveau 1 en modifiant le chromosome de telle sorte qu'on obtient le chromosome utilisé dans l'algorithme IS-NSGA-II afin d'utiliser le même opérateur de croisement.
- La mutation : Pour la mutation, deux cases parmi les m_1 premières colonnes sont choisis aléatoirement et leurs valeurs échangées. Une valeur $\beta \in]0, 1]$ est générée aléatoirement afin de décider si l'ordre de priorité des machines est changé et à quel niveau. Si $\beta \geq 0.5$, on choisit un niveau pour intervertir l'ordre entre deux machines choisies aléatoirement dudit niveau.

6.2.2 MOMSA

On présente dans cette section une approche métaheuristique adaptée de l'algorithme MOMSA introduite par Lin and Ying (2015). MOMSA est une métaheuristique front pseudo-Pareto basée sur le recuit simulé. Comme dans un algorithme recuit simulé, une température initiale et un paramètre de contrôle de la température sont déterminés et une population initiale est engendrée. La population initiale est un mélange d'individus générés aléatoirement et d'individus crées suivant certaines règles d'ordonnancement. On note par P_{al} l'ensemble des individus générés aléatoirement et par P_{cr} l'ensemble des individus non aléatoires. L'ensemble P_{cr} est identique à celui généré par l'algorithme NSGA-II.Les individus générés aléatoirement se font de la même manière que pour l'algorithme NSGA-II. Le codage des solutions est aussi identique.

Nous présentons dans ce qui suit le pseudo-code de l'algorithme.

Algorithm 5 Pseudo code MOMSA

Préciser $T_{init}, T_{min}, N_{pop}, I_{iter}, \alpha$
Générer une population initiale $P_0 = P_{al} \cup P_{cr}$
Evaluer la fonction fitness $(cmax(x), E(x))$ de chaque solution x de la population
Générer l'ensemble front pseudo-Pareto P_s
Calculer $k_1 = \min\{cmax(x)\}, k_2 = \min\{E(x)\}$
Choisir aléatoirement N_{pop} individus à partir de P_s
$T \leftarrow T_{init}, iter \leftarrow 0$
while Critère d'arrêt non satisfait **do**
    $iter \leftarrow iter + 1$
    for $p = 1TOp = N_{pop}$ **do**
        Perturbation de l'individu x_p pour obtenir un individu x'_p
        Calculer la probabilité d'acceptance p_{acc} en fonction de k_1, k_2
        Générer $r \sim U(0, 1)$
        if $r \leq p_{acc}$ **then**
            $x_p \leftarrow x'_p$
            Mettre à jour P_s
        end if
        if x_p domine tous les individus de P_s **then**
            $x_j \leftarrow x_p, \forall j \in \{, \overline{N_{pop}}\}$
        end if
        Mettre à jour les valeur de k_1, K_2
    end for
    if $iter = I_{iter}$ **then**
        $T \leftarrow T \times \alpha$
        $iter \leftarrow 0$
        Choisir aléatoirement N_{pop} individus à partir de P_s
    end if
end while
return L'ensemble P_s

La perturbation appliquée sur les individus choisis est comme suit : Deux valeurs différentes m et m' sont choisis aléatoirement entre 1 et m_1 et une troisième valeur n_0 entre 1 et n, et on inverse les cases $a_{n_0 m}$ et $a_{n_0 m'}$ dans la matrice A du chromosome. De plus, une valeur $\beta \in]0, 1]$ est générée aléatoirement afin de décider si l'ordre de priorité des machines est changé et à quel niveau. Si $\beta \geq 0.5$, on choisit un niveau pour intervertir l'ordre entre deux machines choisies aléatoirement dudit niveau.

6.3 Application

On considère un problème d'ordonnancement flowshop hybride à deux niveaux. Chaque niveau l contient m_l machines parallèles. Le nombre de machines actives simultanément est limité au premier étage. Les tâches ont un temps d'exécution fixe au premier niveau et déteriorable au second étage.

6.3.1 Exemple

On considère dans cet exemple 15 tâches à exécuter sur deux étages. Le premier étage contient 8 machines et au deuxième étage 6 machines.

La table 6.1 représente les valeurs des temps d'exécution de chaque tâche par niveau.

		Tâches														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Temps d'exécution niveau 1														
machines	1	2	8	3	2	6	3	4	2	8	7	10	8	3	5	4
	2	1	7	4	7	6	3	1	5	1	9	10	4	6	8	3
	3	6	3	4	8	4	4	4	7	4	1	4	5	2	2	5
	4	5	4	3	10	3	5	9	6	3	4	8	7	10	3	7
	5	6	6	5	6	7	5	8	1	7	10	7	7	4	5	3
	6	8	3	2	7	8	7	7	2	10	7	9	1	1	8	8
	7	9	7	2	2	3	2	9	8	1	3	4	4	10	9	8
	8	4	6	3	6	7	8	4	3	4	9	5	6	4	10	1
		Temps d'exécution niveau 2														
machines	1	6	7	4	3	5	6	5	4	9	2	2	5	8	6	7
	2	6	5	5	5	7	5	9	8	5	10	5	3	9	5	5
	3	9	6	10	3	3	9	2	9	10	3	3	3	1	5	2
	4	10	5	5	3	4	8	4	7	10	7	8	4	2	4	5
	5	5	1	6	1	7	10	5	1	2	8	7	7	1	1	1
	6	2	5	2	5	1	5	6	7	6	5	2	4	4	5	3

TABLE 6.1: Temps d'exécution des tâches

La table **??** représente la consommation énergétique par unité de temps des machines pour chaque niveau.

Consommation énergétique niveau 1								
Machines	1	2	3	4	5	6	7	8
Consommation énergétique	3	1	7	2	5	3	8	1
Consommation énergétique niveau 2								
Machines	1	2	3	4	5	6		
Consommation énergétique	9	3	9	5	2	10		

TABLE 6.2: Consommation énergétique niveau 1 et 2

La table 6.3 représente le début de déterioration de chaque tâche ainsi que le taux de déterioration.

Tâches	Début de déterioration	Taux de déterioration
1	1	0.9
2	4	0.8
3	2	0.1
4	3	0.7
5	6	0.4
6	5	0.9
7	2	0.5
8	1	0.1
9	4	0.7
10	1	0.5
11	6	0.2
12	6	0.4
13	3	0.3
14	1	0.2
15	4	0.6

TABLE 6.3: Consommation énergétique et taux de détérioration machine

La table 6.4 suivant représente un front pseudo-Pareto obtenu par l'algorithme MOMSA (taille de la population SL=40, HL=10, nombre d'itération 3, $Tmax = 1000, Tmin = 100, \alpha = 0.5$:

solution	Makespan	Consommation énergétique	Ordonnancement
Solution A	18	460	$M_{1,8}$ (0-6) $T_{11} \rightarrow T_{15}$ $M_{1,4}$ (0-3) T_3 $M_{1,5}$ (0-6) $T_6 \rightarrow T_8$ $M_{1,6}$ (0-5) $T_2 \rightarrow T_{12} \rightarrow T_{13}$ $M_{1,1}$ (3-10) $T_1 \rightarrow T_{14}$ $M_{1,7}$ (5-8) $T_4 \rightarrow T_9$ $M_{1,3}$(6-7) T_{10} $M_{1,2}$ (6-13) $T_5 \rightarrow T_7$ $M_{2,6}$(3-15) $T_3 \rightarrow T_1 \rightarrow T_{15} \rightarrow T_{14}$ $M_{2,3}$ (3-15) $T_2 \rightarrow T_5$ $M_{2,2}$ (4-17) $T_{12} \rightarrow T_{10}$ $M_{2,4}$ (5-13) T_{11} $M_{2,1}$ (5-11) T_6 $M_{2,5}$ (5-18) $T_{13} \rightarrow T_8 \rightarrow T_4 \rightarrow T_9 \rightarrow T_7$

TABLE 6.4: Front pseudo-Pareto MOMSA

La figure 6.3.1 représente l'ordonnancement de la solution A.

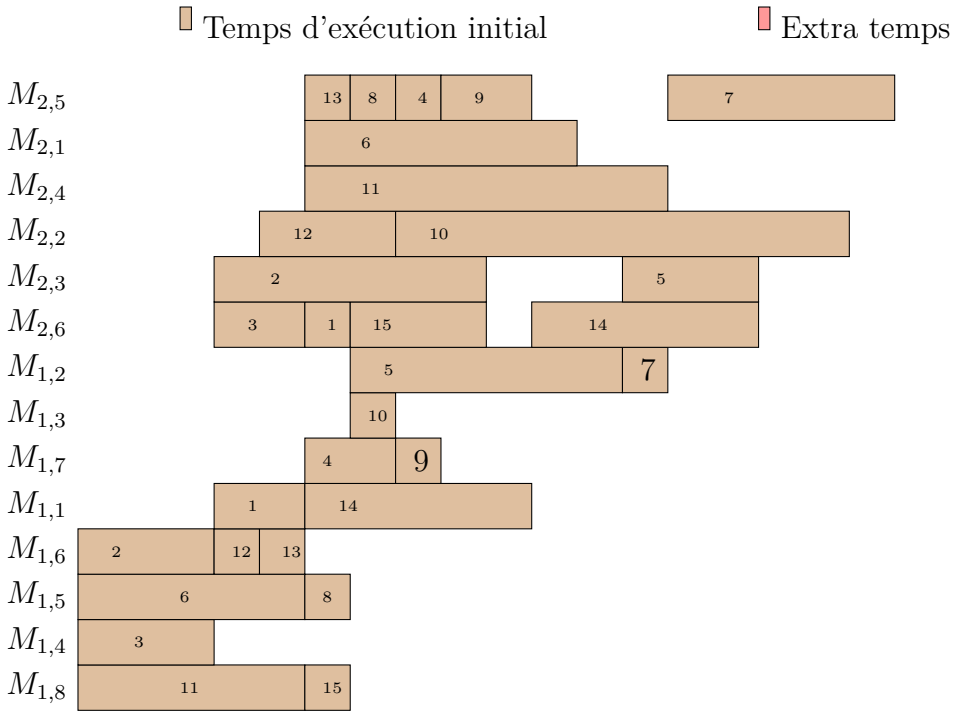


FIGURE 6.1 – ordonnancement de la solution A

La figure 6.3.1 représente la consommation énergétique de la solution A.

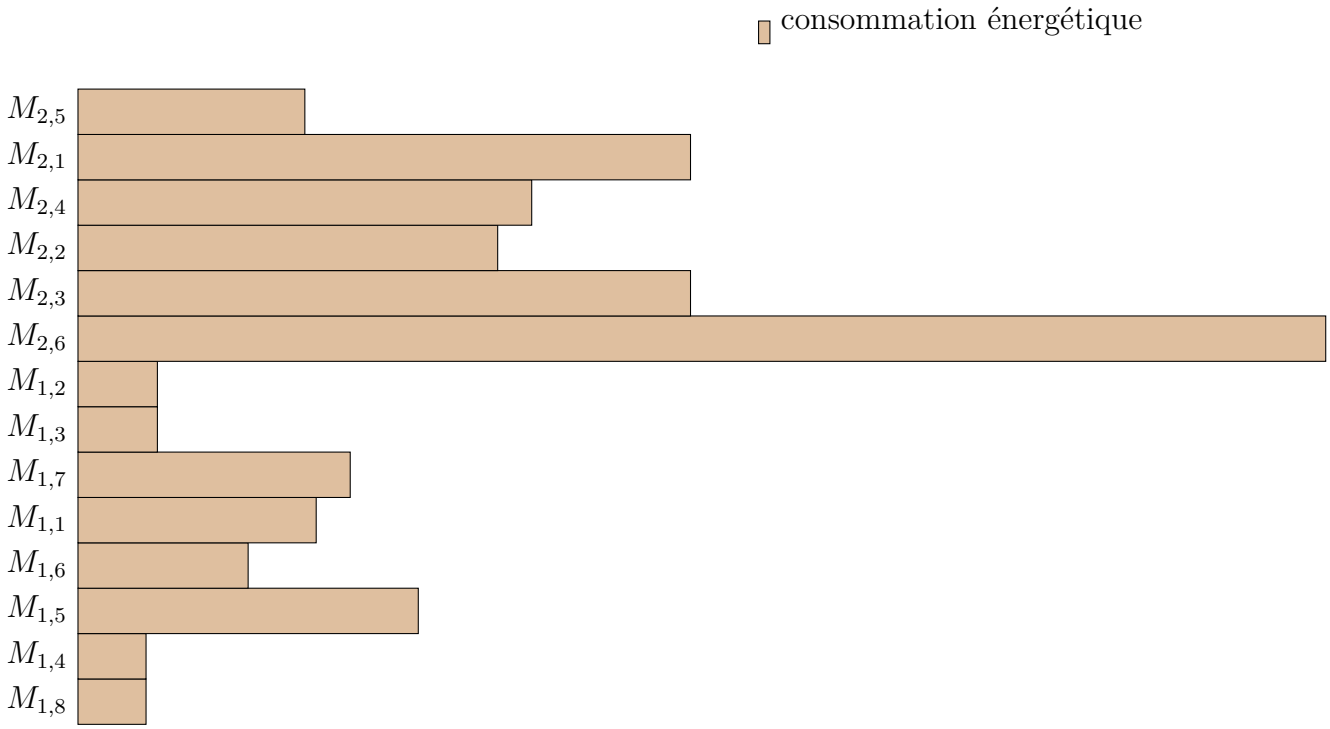


FIGURE 6.2 – Consommation énergétique de la solution A

A partir des figures et , on remarque que les machines $M_{1,8}$ et $M_{1,4}$ ont consommé dans cet ordonnancement la même quantité d'énergie mais le temps de fin d'exécution des tâches sur la machine $M_{1,4}$ est inférieur à celui obtenu dans la machine $M_{1,8}$. On remarque aussi que le temps de fin d'exécution des tâches de la machine $M_{2,4}$ est inférieur à celui de la machine $M_{2,1}$ mais la consommation d'énergie de cette dernière est inférieure. Ce qui démontre que les deux objectifs sont

conflictuels.

Dans la table 6.5, on présente un front pseudo-Pareto de l'algorithme NSGA-II.

solution	Makespan	Consommation énergétique	Ordonnancement
Solution 1	15	536	$M_{1,8}$ (0-9) $T_6 \rightarrow T_{15}$ $M_{1,1}$ (0-2) T_4 $M_{1,6}$ (0-4) $T_{12} \rightarrow T_{13} \rightarrow T_3$ $M_{1,2}$ (0-3) $T_1 \rightarrow T_7 \rightarrow T_9$ $M_{1,5}$ (2-3) T_8 $M_{1,3}$ (3-13) $T_{10} \rightarrow T_{14} \rightarrow T_2 \rightarrow T_{11}$ $M_{1,4}$(3-6) T_5 $M_{2,6}$(1-15) $T_{12} \rightarrow T_{10} \rightarrow T_{11}$ $M_{2,5}$ (1-14) $T_1 \rightarrow T_{14} \rightarrow T_5$ $M_{2,4}$ (2-15) $T_4 \rightarrow T_3 \rightarrow T_2$
-			$M_{2,3}$ (2-12) $T_{13} \rightarrow T_8$ $M_{2,1}$ (2-14) $T_7 \rightarrow T_6$ $M_{2,2}$ (3-14) $T_9 \rightarrow T_{15}$
Solution 2	16	515.3	$M_{1,2}$ (0-3) $T_1 \rightarrow T_7 \rightarrow T_9$ $M_{1,5}$ (0-1) T_8 $M_{1,4}$ (0-3) T_5 $M_{1,1}$ (0-2) T_4 $M_{1,8}$ (1-2) T_{15} $M_{1,6}$ (2-6) $T_{12} \rightarrow T_{13} \rightarrow T_3$ $M_{1,3}$(2-12) $T_{10} \rightarrow T_{14} \rightarrow T_2 \rightarrow T_{11}$ $M_{1,7}$ (3-5) T_6 $M_{2,5}$(1-9.4) $T_1 \rightarrow T_{13} \rightarrow T_{14} \rightarrow T_2$ $M_{2,1}$ (1-10) $T_8 \rightarrow T_{12}$ $M_{2,4}$ (2-14.5) $T_7 \rightarrow T_{10}$ $M_{2,2}$ (2-12) $T_4 \rightarrow T_3$ $M_{2,3}$ (2-16) $T_{15} \rightarrow T_5 \rightarrow T_6$ $M_{2,6}$ (3-14) $T_9 \rightarrow T_{11}$
Solution 3	20	491.88	$M_{1,2}$ (0-3) $T_1 \rightarrow T_7 \rightarrow T_9$ $M_{1,4}$ (0-3) T_5 $M_{1,1}$ (0-2) T_4 $M_{1,8}$ (0-9) $T_6 \rightarrow T_{15}$ $M_{1,6}$ (2-6) $T_{12} \rightarrow T_{13} \rightarrow T_3$ $M_{1,3}$ (3-13) $T_{10} \rightarrow T_{14} \rightarrow T_2 \rightarrow T_{11}$ $M_{1,5}$(3-4) T_8 $M_{2,5}$(1-20) $T_1 \rightarrow T_8 \rightarrow T_{14} \rightarrow T_{11}$ $M_{2,6}$ (2-14) $T_7 \rightarrow T_2$ $M_{2,4}$ (2-12) $T_4 \rightarrow T_{13} \rightarrow T_3$ $M_{2,2}$ (3-14) $T_9 \rightarrow T_{15}$ $M_{2,1}$ (3-14) $T_5 \rightarrow T_6$ $M_{2,3}$ (3-10) $T_{12} \rightarrow T_{10}$
Solution 4	18	500.8	$M_{1,6}$ (0-4) $T_{12} \rightarrow T_{13} \rightarrow T_3$ $M_{1,5}$ (0-1) T_8 $M_{1,4}$ (0-3) T_5 $M_{1,8}$ (0-9) $T_6 \rightarrow T_{15}$ $M_{1,1}$ (1-3) T_4 $M_{1,2}$ (3-6) $T_1 \rightarrow T_7 \rightarrow T_9$

			$M_{1,3}$(3-13) $T_{10} \rightarrow T_{14} \rightarrow T_2 \rightarrow T_{11}$ $M_{2,6}$(1-5) T_{12} $M_{2,5}$ (1-11) $T_8 \rightarrow T_1 \rightarrow T_{15} \rightarrow T_2$ $M_{2,3}$ (2-12.2) $T_{13} \rightarrow T_{14}$ $M_{2,2}$ (3-18) $T_5 \rightarrow T_{11}$ $M_{2,4}$ (3-16) $T_4 \rightarrow T_9$ $M_{2,1}$ (4-14) $T_3 \rightarrow T_6$

TABLE 6.5: Front pseudo-Pareto NSGA-II

La figure 6.3 représente l'ordonnancement de la solution 3.

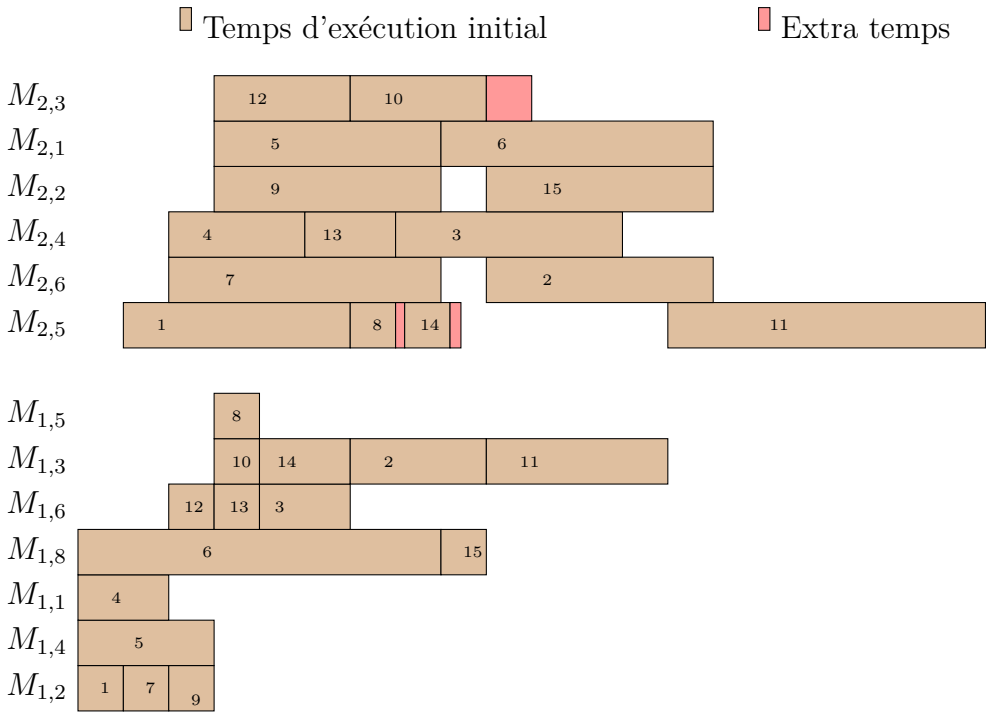


FIGURE 6.3 – Ordonnancement de la solution 3

La figure 6.4 représente la consommation énergétique des machines.

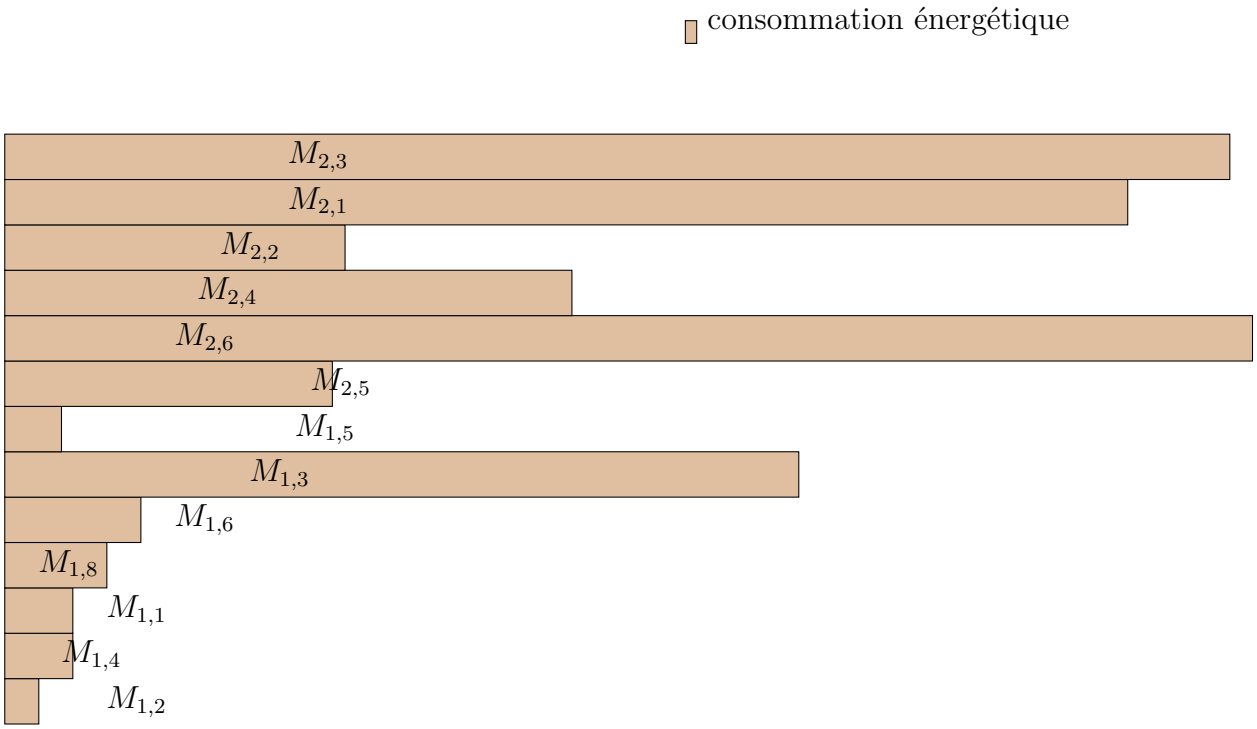


FIGURE 6.4 – Consommation énergétique de la solution 3

On remarque que pour certaines solutions, les machines n'ont pas été toutes utilisées au niveau un.

En comparant les deux fronts pseudo-Pareto, on remarque que la solution obtenue par MOMSA domine toutes les solutions du front pseudo-Pareto de NSGA-II. De plus, en regardant les solutions représentées, on remarque que dans la solution obtenue par Momsa, aucune tâche n'a été traitée après la date de début de déterioration même si le makespan des deux solutions est identique.

6.3.2 Comparaison

Des tests ont été faites afin de comparer les résultat obtenus par les deux algorithmes NSGA-II et MOMSA où les solutions de chaque algorithme ont été classées grâce à TOPSIS. On a considéré des problèmes à petites, moyennes et grandes tailles.

On remarque que l'algorithme NSGA-II donne de meilleurs résultats que l'algorithme MOMSA. En comparant leur temps d'exécution, l'algorithme NSGA-II est plus rapide que MOMSA.

Poids TOPSIS (makespan - Consommation énergétique)	Instances	Makespan et Consommation énergétique de la meilleure solution	
		NSGA-II	MOMSA
	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	20-357
	$8 \times 6 \times 15$	12-403	16-399
	$8 \times 6 \times 20$	19-517.4	24-500
	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-968.2	26-1081.16
	$15 \times 10 \times 50$	27-1541.96	28-1538.4

0.3-0.7

0.5-0.5	$15 \times 10 \times 60$	34-2724.5	47-3038
	$15 \times 10 \times 70$	41-2942.118	61-3353
	$20 \times 15 \times 80$	35.446-3166.302	43-3013.5
	$20 \times 15 \times 90$	39-3862.86	53-4597
	$25 \times 20 \times 90$	29-4009.355	31-3880.81
	$25 \times 20 \times 100$	34-4527.69	45-5821
	$25 \times 20 \times 150$	50.52-6957.586	73-8042
	$30 \times 25 \times 150$		65-7605
	$30 \times 25 \times 200$	51-9968.294	84-10965
	$30 \times 25 \times 250$	66-12426.337	100-14127
	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	18-378.2
	$8 \times 6 \times 15$	12-403	13.1-478.5
	$8 \times 6 \times 20$	19-527.4	24-500
	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-986.2	26-1081.16
	$15 \times 10 \times 50$	27-1541.96	26.4-1602.8
	$15 \times 10 \times 60$	34-2724.5	45-3208
	$15 \times 10 \times 70$	39.6-2906.078	55-3527
	$20 \times 15 \times 80$	35.446-3166.302	39-3271.85
	$20 \times 15 \times 90$	37.52-3974.09	50-4728
	$25 \times 20 \times 90$	29-4009.355	30.691-3979.113
	$25 \times 20 \times 100$	32.33-4646.74	45-5821
	$25 \times 20 \times 150$	48-6714.466	69-8376
	$30 \times 25 \times 150$	40-6267.785	62-7778
	$30 \times 25 \times 200$	51-9968.294	77-11419
	$30 \times 25 \times 250$	66-12426.337	94-14599
0.7-0.3	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	18-378.2
	$8 \times 6 \times 15$	12-403	13.1-478.5
	$8 \times 6 \times 20$	19-527.4	21-571.2
	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-986.2	24-1178.88
	$15 \times 10 \times 50$	27-1541.96	26.4-1602.8
	$15 \times 10 \times 60$	34-2724.5	43-3369.324
	$15 \times 10 \times 70$	39.6-2906.078	55-3527
	$20 \times 15 \times 80$	35.446-3166.302	35.82-3460.456
	$20 \times 15 \times 90$	37.52-3974.09	50-4728
	$25 \times 20 \times 90$	29-4009.355	29-4113.603
	$25 \times 20 \times 100$	32.08-4900.94	45-5821
	$25 \times 20 \times 150$	48-6714.466	69-8376
	$30 \times 25 \times 150$	40-6267.785	60-8055
	$30 \times 25 \times 200$	51-9968.294	77-11419
	$30 \times 25 \times 250$	66-12426.337	94-14599

TABLE 6.6: Performances des algorithmes

Conclusion et perspectives

Dans cette thèse, on a présenté un état de l'art sur les problèmes d'ordonnancement considérant la gestion de l'énergie comme objectif ou contrainte.

On a établi un état de l'art sur les problèmes d'ordonnancement considérant des tâches dont le temps d'exécution varie selon un facteur indépendant de la machine.

on a présenté un problème d'ordonnancement bi-objectif de tâches déteriorables sur un ensemble de machines générales. On cherche à minimiser simultanément le makespan et la consommation énergétique. On a présenté un modèle mathématique et un algorithme énumératif. Le temps d'exécution de cet algorithme croit avec la taille de l'instance, on a choisi de travailler avec des métaheuristiques. On a présenté quelques métaheuristiques basées sur le NSGA-II (IS-NSGA-II, OSMS-NSGA-II, NSGA-II hybride) dont on a comparé l'efficacité en utilisant la méthode de comparaison TOPSIS. Les résultats obtenus montrent que la métaheuristique NSGA-II hybride converge plus rapidement que les deux autres.

On a proposé un problème d'ordonnancement bi-objectif dans un système flow shop hybride avec des tâches déteriorables. Les objectif sont la minimisation du makespan et de la consommation énergétique. On a proposé un modèle mathématique. On a utilisé des métaheuristiques basés sur le recuit simulé et le NSGA-II. On a comparé les deux approches.

Comme perspective, on se propose d'améliorer la performances des méthodes utilisées. Il est intéressant d'étudier ces problèmes en considérant d'autres critères d'optimalité ou d'étudier la consommation énergétique en tant que contrainte.

Annexe A

Scheduling and energy consumption

Dans ce chapitre, on donne l'essentiel des résultats obtenus aux chapitres 4 et 5 en version anglaise.

A.1 Multiobjective approach for deteriorating jobs scheduling for a sustainable manufacturing system

The reader can find the full version in (Tigane et al., 2019).

A.1.1 Introduction

In all scheduling problems with energy concerns we encountered, processing times of the jobs were supposed to be constant or depending on the machine's speed.

However, there are models where the processing time varies over time independently of the machine's speed.

In such models, the processing time can be an increasing function of the starting time as in the scheduling of steel rolling mills or in medical procedures under health conditions. It can also be a decreasing function such as in military applications in which a task consists of destroying an approaching aerial object, or in agriculture where harvesting may exhibit decreasing processing times as the crop dries.

In this paper, we treat a scheduling problem with deteriorating jobs that combines the two objectives of minimizing the makespan and the energy consumption.
We present in the next section the considered problem and the associated mathematical formulation. Then, we present an enumerative algorithm and two metaheuristics based on the NSGA-II algorithm for multi-objective problems. Finally, we discuss the performance of the algorithms.

A.1.2 Problem formulation

The considered study is inspired by the hot rolling process. It deals with the scheduling problem of hot slabs production jobs on a hot rolling production lines.

In this study, we consider an offline scheduling problem on m unrelated machines. An instance of the scheduling problem consists of n independent and non-preemptive and non-migratory jobs available at time $s_0 = 0$ and, where each job is defined by a start time dependent processing times. Idleness of machines is not allowed.

The job's processing time is a non-decreasing, linear function given by

$$p_{ij} = p_{ij}^0 + \alpha_i s_{ij} \tag{A.1}$$

Where p_{ij}^0 is the basic processing time of job i in machine j, $\alpha_{ij} > 0$ is the deterioration rate, and $s_{ij} \geq s_0$ is the start time of job i on machine j. The basic processing times are supposed to be integers and the growth rates are supposed to be real numbers. We assume that jobs stop deteriorating as soon as the processing on the machine begins. The objective is to minimize the makespan and the total energy consumption TEC.
The completion time of job i on machine j is $C_{ij} = s_{ij} + p_{ij}$. Since there is no interruption during the schedule, the completion time of the jobs processed in machine j is then $C_j = \sum_{i \in J_j} p_{ij}$, where J_j is the set of the jobs processed on machine j.
We suppose that machine j consumes δ_j energy per unit of time during the process. Thus, the energy consumed for the processing of job i is $p_{ij}\delta_j$, and that energy increases by the rate θ_j over time. The total energy consumed for the processing of the job is given by the following equation :

$$E_{ij} = p_{ij}\delta_j + \theta_j s_{ij} \tag{A.2}$$

Then, the total energy consumption by a machine j is :

$$E_j = \sum_{i=1}^{n} (p_{ij}\delta_j + \theta_j s_{ij}) \tag{A.3}$$

Since the conditions about machines and jobs are quite the same as in Mazdeh et al. (2010), the formulation of the constraints and notation for the decision variables are adapted from (Li et al., 2014, Mazdeh et al., 2010) and repeated here for the reader's convenience.
The two objective functions considered in the studied problem are expressed as follows :

1. The minimization of the maximum completion time C_{max} :

$$\min C_{max} = \min \max_{j \in \{\overline{1,m}\}} \{C_{max}^{(j)}\} = \min \max_{j \in \{\overline{1,m}\}} \{\sum_{i=1}^{n} p_{ij} x_{ij}\} \tag{A.4}$$

2. The minimization of the total energy consumption TEC :

$$\min TEC = \min \sum_{j=1}^{m} E_j = \min \sum_{j=1}^{m} (\sum_{i=1}^{n} (p_{ij}\delta_j + \theta_j s_{ij}) x_{ij}) \tag{A.5}$$

The constraints are expressed as follows :

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in \{\overline{1,n}\} \tag{A.6}$$

$$(s_{ij} + p_{ij})x_{ij} = s_{kj}y_{ikj} \quad \forall i,k \in \{\overline{1,n}\}, k \neq i, j \in \{\overline{1,m}\} \tag{A.7}$$

$$\sum_{k=1}^{n} y_{0kj} \leq 1 \quad \forall j \in \{\overline{1,m}\} \tag{A.8}$$

$$\sum_{k=1,k\neq i}^{n} y_{ikj} \leq x_{ij} \quad \forall i \in \{\overline{1,n}\}, j \in \{\overline{1,m}\} \tag{A.9}$$

$$\sum_{i=0,i\neq k}^{n} y_{ikj} = x_{kj} \quad \forall k \in \{\overline{1,n}\}, j \in \{\overline{1,m}\} \tag{A.10}$$

$$\sum_{i=0,i\neq k}^{n} \sum_{j=1}^{m} y_{ikj} = 1, \quad \forall k \in \{\overline{1,n}\} \tag{A.11}$$

$$\sum_{i=1}^{n} p_{ij}x_{ij} \leq C_{max} \quad \forall j \in \{\overline{1,m}\} \tag{A.12}$$

$$s_{0j} = 0 \quad \forall j \in \{\overline{1,m}\} \tag{A.13}$$

$$s_{ij} \geq 0, C_{ij} \geq s_{ij} \quad \forall i \in \{\overline{1,n}\}, j \in \{\overline{1,m}\} \tag{A.14}$$

$$x_{ij} \in \{0,1\}, y_{ikj} \in \{0,1\} \tag{A.15}$$

Objective 1 is to minimize the makespan. Objective 2 is to minimize the total energy consumption.

Constraint A.6 states that each job is processed by exactly one machine.

Constraint A.7 states that the completion time of a job processed in a machine is equal to the starting time of the job following it, which means that there is no idle time.

For each dummy job 0 allocated to a machine, only one real job can immediately follow it (Constraint A.8). This also means that the scheduler can decide whether or not to process jobs on a machine if it leads to an optimal schedule.

Constraints A.9 and A.10 show that n jobs are assigned over m machines and if job i is immediately followed by job k on machine j, then both jobs i and k belong to m (i.e. no-migration between machines).

Constraint A.11 means that only one job can immediately precede a job (including dummy jobs) and jobs are processed only by one machine (no-parallelism of jobs).

Constraint A.12 means that the total completion time in a machine is always less than the makespan.

Constraint A.13 states that the start time of a dummy job is $s_0 = 0$ for any machine.

Constraint A.14 states that the start time is always positive and the completion time of a job is greater than its start time.

A.1.3 Algorithmic approach

We develop a pseudo exact algorithm that provides the near-Pareto front of all the possible solutions, where the order of the jobs in a machine fulfills the optimal policy provided by Browne and Yechiali (1990) for the makespan minimization of deteriotating jobs on a single machine. **Algorithm 6** details the pseudo code of the pseudo exact algorithm.

Algorithm 6 Pseudo-Exact algorithm pseudo-code

```
1: procedure SCHEDULE(n, m)
2:     k ← 1
3:     F ← ∅                                          ▷ Gives the near-Pareto set
4:     while k <= m do
5:         Get all the combinations without repetition of k machines among m
6:         for Each combination obtained do
7:             Get all the permutations possible
8:             for Each permutation obtained do
9:                 Get all the partitions of the set of n jobs in k subsets, the
   allocation of the subsets to machines follow the order of the permutation
   and jobs are sorted in machines in the non-decreasing order of p_ij^0/α_i. Each
   partition corresponds to a S
10:                Calculate the corresponding C_max and E of a schedule S
11:                if F = ∅ then
12:                    F ← S
13:                else
14:                    for Each schedule T ∈ F do
15:                        if C_max, E of S are better than those of T then
16:                            F ← S
17:                            Remove T from F
18:                        else if C_max, E of S are worse than those of T then
19:                            Keep T in F
20:                        else
21:                            F ← S
22:                        end if
23:                    end for
24:                end if
25:            end for
26:        end for
27:        k ← k + 1
28:    end whilereturn F
29: end procedure
```

Since the calculus time of the algorithm increases with the increase of the instance, it is more practical to use a metaheuristic to solve the problem. Therefore, we present an NSGA-II based algorithm. We call it OSMS-NSGA-II.

We propose also an integrated scheduling based NSGA-II approach (IS–NSGA-II) that takes into account all the possible allocations, and order of jobs in different machines.

Algorithm 7 detail respectively the pseudo-codes of the OSMS–NSGA-II approach.

Algorithm 9 details the pseudo-code of the IS–NSGA-II approach.

Algorithm 7 OSMS–NSGA-II pseudo-code

Generate a random initial population P_0 of size l
$t = 1$
Order the jobs i in a machine j in a non-decreasing order of p_{ij}^0/α_i
Evaluate the fitness function of each solution
Sort solutions by non domination and crowding distance to obtain the non-dominated groups
while Stopping criterion is not satisfied **do**
    Create the children's group C of size l by crossover and mutation
    Order the jobs i in a machine j in a non-decreasing order of p_{ij}^0/α_i
    Evaluate the fitness function of each solution of the population $P \cup C$
    Sort solutions by non domination and crowding distance to obtain the non-dominated groups
    Rank in solutions in the population P_t
    $t = t + 1$
end while

Algorithm 9 IS–NSGA-II pseudo-code

Generate a random initial population P_0 of size l
$t = 1$
Evaluate the fitness function of each solution
Sort solutions by non domination and crowding distance to obtain the non-dominated groups
while Stopping criterion is not satisfied **do**
    Create the children's group C of size l by crossover and mutation
    Evaluate the fitness function of each solution of the population $P \cup C$
    Sort solutions by non domination and crowding distance to obtain the non-dominated groups
    Rank in solutions in the population P_t
    $t = t + 1$
end whilereturn The non-dominated group of rank 1

A.1.4 Comparison

Several computational experiments were made to evaluate the performance of the OSMS NSGA-II algorithm The basic processing times and energy costs were supposed non-negative integers and the deterioration rates and growth rate non-negative real numbers. The algorithm was coded in the Java Netbeans environment and the experiments performed on a computer with a 2.20 GHz Intel core i5-5200U CPU and 8 GB RAM under Windows 10.

We test the exact algorithm on several instances in order to evaluate its performance in terms of execution time (Table A.1).

		m		
		3	4	5
n	5	<1s	1s	1s
	8	1 s	1 s	2 s
	10	1 s	4 s	46 s

	12	3 s	57 s	22 min 24 s
	15	7min 51sec	113 min 20 sec	2911 min 59 s

TABLE A.1: Time execution of the exact algorithm

In Table A.1, we observe that the execution time of the exact algorithm seems to increase rapidly. The tests performed on the OSMS NSGA-II algorithm show that we can obtain good approximate near-Pareto fronts within minutes.

To enhance the results of the proposed OSMS-NSGA-II and IS-NSGA-II approaches, we consider a improved NSGA-II based approach described as follows :
In this new approach, the first population is composed of solutions randomly generated and selected solutions.

The selected solutions are obtained by using some sequencing rules, as follows :

1. First rule :
   In each machine, the jobs are sorted by the non decreasing order of p_{ij}^0/α_i and then, at each time t, we choose the first available machine and the first job available in the ranking on this machine, and so on.
2. Second rule :
   Each job is assigned to the machine where the basic processing time is the smallest. Then, in each machine, jobs are sorted by the non decreasing order of p_{ij}^0/α_i.
3. Third rule :
   In each machine, the jobs are sorted by the non decreasing order of p_{ij}^0/α_i and then, at each time t, we choose the fastest machine available and the first job available in the ranking on this machine, and so on.

Then, the NSGA-II is used to find the optimal near-Pareto front without any scheduling rules.

Computational experiments have been performed on small, average and large instances to compare the performance of the three proposed approaches.

We observe that with the increase of the instance, the improved approach seems to give better results than the two other approaches.
Indeed, the improved approach obtained the best solution among the three proposed approaches for 34 instances out of 54 (green cells in Table A.2). In the same time, it gives a very good solution, non-dominated by solutions of the other solutions for 20 instances (yellow cells in Table A.2).

TOPSIS weights (makespan - energy consumption)	Instances	Makespan and energy consumption of the best solution		
		OSMS-NSGA-II	IS-NSGA-II	Improved approach
	3×10	21.68-29.468	21.68-30.068	21.68-29.468
	3×20	66.364-177.4	88.306-206.543	67.918-168.54
	5×20	22.16-70.7	27.164-69.979	19.751-60.698
	5×30	45.506-150.21	46.728-158.037	36.838-108.428
	8×20	8-28.394	7.68-28.246	7.42-29.358
	8×30	14.696-68.206	17.45-61.442	12.504-57.306
	10×40	20.209-110.797	12.952-77.863	12.952-77.863

	10×50	35.804-176.515	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	8.001-47.561	5.64-45.604
	15×50	9.132-72.646	10.54-74.728	9.152-65.83
	20×50	6.2-49.925	8-55.77	4.99-53.695
	20×60	8.08-79.851	8.3-82.482	7.356-71.54
	15×120	57.768-714.32	64.07-740.675	37.844-608.225
	15×150	119.435-1501.755	129.803-1621.277	61.57-1218.283
	20×150	55.57-803.388	60.748-920.236	27.801-627.554
	20×180	95.12-1621.69	114.977-1764.36	48.752-1175.453
	25×180	51.335-852.61	51.495-986.338	27.425-700.327
	25×200	71.458-1219.765	82.346-1479.756	38.623-1013.234
0.5-0.5	3×10	20.16-32.204	21.68-30.068	18.64-32.8
	3×20	66.364-177.4	84.616-208.56	63.84-172.977
	5×20	22.16-70.7	28.192-74.472	19.751-60.698
	5×30	45.506-150.21	42.254-167.155	36.838-108.428
	8×20	7.6-29.734	7.68-28.246	7.42-29.358
	8×30	14.696-58.206	16.02-64.831	12.334-57.629
	10×40	20.209-110.797	12.952-77.863	12.952-77.863
	10×50	33.473-184.736	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	6.96-51.117	5.64-45.722
	15×50	9.132-72.646	10.54-74.728	7.816-73.48
	20×50	6.2-49.925	6.77-62.162	4.99-53.695
	20×60	7.6-81.996	8.3-82.482	7.356-71.54
	15×120	56.705-722.112	64.07-740.675	37.083-615.285
	15×150	119.435-1501.755	124.146-1655.302	61.57-1218.283
	20×150	51.99-839.691	60.748-920.236	27.801-627.554
	20×180	91.347-1686.64	114.977-1764.36	47.431-1205.122
	25×180	48.864-863.3	50.196-988.87	26.927-710.223
	25×200	70.822-1318.538	78.803-1555.967	38.178-1017.288
0.7-0.3	3×10	20.16-32.204	20.64-33.924	18.64-32.8
	3×20	66.364-177.4	84.616-208.56	63.84-172.977
	5×20	22.16-70.7	24.53-74.521	19.751-60.698
	5×30	41.514-170.897	42.254-167.155	36.838-108.428
	8×20	7.6-29.734	7.68-28.246	7.18-30.846
	8×30	14.696-58.206	16.02-64.831	12.334-57.629
	10×40	20.209-110.797	12.952-77.863	12.952-77.863
	10×50	32.622-190.133	32.648-230.121	17.816-124.307
	15×40	5.96-43.05	6.96-51.117	5.64-45.722
	15×50	9.132-72.646	9.89-78.002	7.816-73.48
	20×50	6.2-49.925	6.77-62.162	4.99-53.695
	20×60	7.6-81.996	8.02-85.537	7.12-73.86
	15×120	56.705-722.122	62.764-773.6	36.427-629.361
	15×150	113.57-1685.15	121.9-1669.534	61.57-1218.283
	20×150	50.234-887.036	58.837-960.626	27.801-627.554
	20×180	90.19-1731.75	106.153-1984.218	46.535-1205.47
	25×180	47.18-920.152	50.196-988.87	26.039-722.354
	25×200	70.265-1324.357	78.803-1555.967	35.772-1095.143

TABLE A.2: Improved approach performances

A.2 An energy problem in a flowshop hybrid scheduling problem

A.2.1 The scheduling problem

We consider a hybrid flow shop scheduling of s-stages. In each stage $k \in \{1, \ldots, s\}$, we have m_k parallel machines. The instance is a set of n jobs where each job is composed of s operations that are scheduled on each stage following the same order, which is stage 1, stage 2 and so on. The jobs are independent, non-preemptive and non migratory. The execution time of the operations are deterministics in some stages and start time dependent on others. The execution time function is formulated as follows : Let t_{ik} be the start time of O_i on stage k, $c_{i(k-1)}$ the finish time of operation O_i on stage k, D_{ik} the time where the operation O_i begins to deteriorate in stage k. We have, by convention $c_{i(-1)} = 0$. So the execution time function is the following :

$$p_{ijk} = \begin{cases} p^0_{ijk} & if\ t_{ik} \leq D_{ik} + c_{i(k-1)} \\ p^0_{ijk} + \alpha_{ik}(t_{ik} - D_{ik} - c_{i(k-1)}) & if\ t_{ik} > D_{ik} + c_{i(k-1)} \end{cases} \tag{A.16}$$

where p^0_{ijk} is the basic execution time of O_i in machine j and α_{ik} is the deterioration rate. D_{ik} is infinite when time execution is deterministic. We suppose that we cannot have all the machines active simultaneously in some stages.

Let C_{max} be the makespan, e_{jk} the energy consumed by an unit of time in active mode by the machine j at stage k. The energy when the machine is idle is supposed negligeable. Let E_{jk} the total energy consumption of the machine j and TEC the total energy consumed on all stages by all the machine, the calculation of C_{max} and TEC are given by the following formulas :

$$\min C_{max} = \min \max_{i \in \{1,\ldots,n\}} C_{is} \tag{A.17}$$

$$\min TEC = \min \sum_{k=1}^{s} \sum_{j=1}^{m_k} E_{jk} = \min \sum_{k=1}^{s} \sum_{j=1}^{m_k} e_{jk} \sum_{i=1}^{n} p_{ijk} \tag{A.18}$$

A.2.2 The mathematical formulation

We first, give the paremeters needed for our mathematical formulation :

- i, l indexes of jobs, $i, l \in \{1, \ldots, n\}$,
- k index of stage, $k \in \{1, \ldots, s\}$,
- m_k number of machines on stage k,
- j index of machines on stage k, $j \in \{1, \ldots, m_k\}$,
- v_k maximum number of active machines simultaneously on stage k

and the variables

- x_{ijk} is a binary variable where $x_{ilk} = 1$ if operation O_i is executed by machine j in stage k and 0 otherwise,
- y^k_{ilj} is a binary variable where $y^k_{ilj} = 1$ if operation O_l follows immediately the operation O_i on the machine j at stage k and 0 otherwise,

- t is the instant present,
- z_{il}^k is a binary variable where $z_{il}^k = 1$ if $[t_{ik}, c_{ik}] \cap [t_{lk}, c_{lk}] \neq \emptyset$, 0 otherwise,
- C_{ik} completion time of operation O_{ik} at stage k.

We can now express our mathematical formulation as follows

$$\sum_{j=1}^{m_k} x_{ijk} = 1 \quad \forall i \in \{1, \ldots, n\}, k \in \{1, \ldots, s\} \tag{A.19}$$

$$\sum_{i=1, i\neq l}^{n} z_{il}^k \leq v_k \quad \forall k \in \{1, \ldots, s\},\ l \in \{1, \ldots, n\} \tag{A.20}$$

$$C_{i1} \geqslant \sum_{j=1}^{m_k} x_{ij1} p_{ij1} \quad \forall i \in \{1, \ldots, n\}, k \in \{1, \ldots, s\} \tag{A.21}$$

$$C_{ik} - c_{i(k-1)} \geqslant \sum_{j=1}^{m_k} x_{ijk} p_{ijk} \quad \forall i \in \{1, \ldots, n\}, k \in \{2, \ldots, s\} \tag{A.22}$$

$$c_{lk} \geqslant c_{ik} + \sum_{j=1}^{m_k} p_{ijk} y_{ilj}^k + \left((\sum_{j=1}^{m_k} y_{ilj}^k) - 1 \right) .B \quad \forall i, l \in \{1, \ldots, n\}, k \in \{1, \ldots, s\} \tag{A.23}$$

$$\sum_{i=0, l\neq i}^{n} y_{ilj}^k = x_{ljk} \quad \forall l \in \{1, \ldots, n\}, j \in \{1, \ldots, m_k\}, k \in \{1, \ldots, s\} \tag{A.24}$$

$$\sum_{l=1, i\neq l}^{n+1} y_{ilj}^k = x_{ijk} \quad \forall i \in \{1, \ldots, n\}, k \in \{1, \ldots, s\}, j \in \{1, \ldots, m_k\} \tag{A.25}$$

$$\sum_{i=0, i\neq l}^{n} \sum_{j=1}^{m_k} y_{ilj}^k = 1 \quad \forall j \in \{1, \ldots, n\}, k \in \{1, \ldots, s\} \tag{A.26}$$

$$z_{il}^k = 1 - \min \left[\min \left(\left| E \left(\frac{c_{lk} - c_{ik}}{\sum_{j=1}^{n} x_{ljk}} \right) \right|, \left| E \left(\frac{c_{lk} - c_{ik}}{\sum_{j=1}^{n} x_{ljk}} \right) \right| \right), 1 \right] \tag{A.27}$$

$$C_{max} \geq C_{is} \quad \forall i \in \{0, \ldots, n\} \tag{A.28}$$

$$x_{ijk}, y_{ilj}^k, z_{il}^k \in \{0, 1\}, \tag{A.29}$$

The constraints A.19,A.26 express the non parallelism and the non migration of jobs. Contraint A.20 shows that the maximum number of active machines is respected. Constraint A.21 expresses that operations are non preemptives and A.22 shows that we cannot execute an operation O_{ik} until $O_{i(k-1)}$ is finished. A.23 shows that the start time of an operation of a job in a stage is at least equal to the completion time of the operation that precedes her on the same machine. A.24 shows that every job follows immediately at most one job. Constraint A.25 shows that every job is immediately followed by at most one job. Constraint A.28 shows that completion times on the last operations are less than the makespan. The expression of A.27 allows the scheduler to calculate the precise value of z_{il}^k.

The objective is to minimize C_{max} and TEC given by the expressions A.17 and A.18.

A.2.3 The approaches

We propose an algorithm based on MOMSA, a multi-objective metaheuristic introduced by Lin and Ying (2015) and an algorithm based on NSGA-II. We give the pseudo-code of the proposed algorithm based on MOMSA. In each algorithm, the initial population is composed of a set of

solutions created by the scheduler and a set of random solutions.

Algorithm 10 Pseudo code MOMSA

Préciser $T_{init}, T_{min}, N_{pop}, I_{iter}, \alpha$
Generate the initial population $P_0 = P_{ran} \cup P_{cr}$
Evaluate the fitness function $(cmax(x), E(x))$ of each solution x of the population
Generate the front near-Pareto set P_s
Calculate $k_1 = \min\{cmax(x)\}, k_2 = \min\{E(x)\}$
Choose randomly N_{pop} individuals from P_s
$T \leftarrow T_{init}, iter \leftarrow 0$
while Stop criterion non satisfied **do**
    $iter \leftarrow iter + 1$
    for $p = 1TOp = N_{pop}$ **do**
        Perturbate x_p to obtain x'_p
        Calculate the acceptance probability p_{acc} in terms of k_1, k_2
        Generate $r \sim U(0, 1)$
        if $r \leq p_{acc}$ **then**
            $x_p \leftarrow x'_p$
            Update P_s
        end if
        if x_p dominate all the individuals of P_s **then**
            $x_j \leftarrow x_p, \forall j \in \{, \overline{N_{pop}}\}$
        end if
        Update k_1, K_2
    end for
    if $iter = I_{iter}$ **then**
        $T \leftarrow T \times \alpha$
        $iter \leftarrow 0$
        Choose randomly N_{pop} individuals from P_s
    end if
end whilereturn Set P_s

A.2.4 Comparison

Computational tests were performed on small, medium and big instances to compare the performance of the two approaches. We used the TOPSIS method to order the solutions of the front near-Pareto.

We observed that the NSGA-II approach gives better solutions than the MOMSA and it is much faster than the latter.

TOPSIS wheights (makespan - Energy consumption)	Instances	Makespan and energy consumption of the best solution	
		NSGA-II	Momsa
	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	20-357
	$8 \times 6 \times 15$	12-403	16-399
	$8 \times 6 \times 20$	19-517.4	24-500

0.5-0.5	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-968.2	26-1081.16
	$15 \times 10 \times 50$	27-1541.96	28-1538.4
	$15 \times 10 \times 60$	34-2724.5	47-3038
	$15 \times 10 \times 70$	41-2942.118	61-3353
	$20 \times 15 \times 80$	35.446-3166.302	43-3013.5
	$20 \times 15 \times 90$	39-3862.86	53-4597
	$25 \times 20 \times 90$	29-4009.355	31-3880.81
	$25 \times 20 \times 100$	34-4527.69	45-5821
	$25 \times 20 \times 150$	50.52-6957.586	73-8042
	$30 \times 25 \times 150$		65-7605
	$30 \times 25 \times 200$	51-9968.294	84-10965
	$30 \times 25 \times 250$	66-12426.337	100-14127
	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	18-378.2
	$8 \times 6 \times 15$	12-403	13.1-478.5
	$8 \times 6 \times 20$	19-527.4	24-500
	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-986.2	26-1081.16
	$15 \times 10 \times 50$	27-1541.96	26.4-1602.8
	$15 \times 10 \times 60$	34-2724.5	45-3208
	$15 \times 10 \times 70$	39.6-2906.078	55-3527
	$20 \times 15 \times 80$	35.446-3166.302	39-3271.85
	$20 \times 15 \times 90$	37.52-3974.09	50-4728
	$25 \times 20 \times 90$	29-4009.355	30.691-3979.113
	$25 \times 20 \times 100$	32.33-4646.74	45-5821
	$25 \times 20 \times 150$	48-6714.466	69-8376
	$30 \times 25 \times 150$	40-6267.785	62-7778
	$30 \times 25 \times 200$	51-9968.294	77-11419
	$30 \times 25 \times 250$	66-12426.337	94-14599
0.7-0.3	$7 \times 5 \times 10$	12-254	14-278
	$7 \times 5 \times 15$	14-380.5	18-378.2
	$8 \times 6 \times 15$	12-403	13.1-478.5
	$8 \times 6 \times 20$	19-527.4	21-571.2
	$8 \times 6 \times 25$	21.2-860.48	21.8-890.8
	$10 \times 8 \times 25$	15.4-600.4	20.2-603.02
	$10 \times 8 \times 30$	21-986.2	24-1178.88
	$15 \times 10 \times 50$	27-1541.96	26.4-1602.8
	$15 \times 10 \times 60$	34-2724.5	43-3369.324
	$15 \times 10 \times 70$	39.6-2906.078	55-3527
	$20 \times 15 \times 80$	35.446-3166.302	35.82-3460.456
	$20 \times 15 \times 90$	37.52-3974.09	50-4728
	$25 \times 20 \times 90$	29-4009.355	29-4113.603
	$25 \times 20 \times 100$	32.08-4900.94	45-5821
	$25 \times 20 \times 150$	48-6714.466	69-8376
	$30 \times 25 \times 150$	40-6267.785	60-8055
	$30 \times 25 \times 200$	51-9968.294	77-11419

	$30 \times 25 \times 250$	66-12426.337	94-14599

TABLE A.3: Performances des algorithmes

Bibliographie

M. Afzalirad and J. Rezaeian. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers and Industrial Engineering*, 98 :40 – 52, 2016. 1.1

K. Alaykyran, O. Engin, and A. Doyen. Using ant colony optimization to solve hybrid flow shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 35(5) : 541–550, 2007. 1.3

S. Albers and A. Antoniadis. Race to idle : New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2), February 2014. ISSN 1549-6325. 4.2.1, 4.2.2

S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3(4), November 2007. 4.2.1

S. Albers, F. Muller, and S. Schmelzer. Speed scaling on parallel processors. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 289–298, 2007. 4.2.2

S. Albers, E. Bampis, D. Letsios, G. Lucarelli, and R. Stotz. Scheduling on power-heterogeneous processors. In *Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico*, pages 41–54, 2016. 4.2.2

I. M. Alharkan. Algorithms for sequencing and scheduling, 2010. 1

B. Alidaee and F. Landram. Scheduling deteriorating jobs on a single machine to minimize the maximum processing times. *International journal of systems science*, 27(5) :507–510, 1996. 3.1.1

B. Alidaee and N. K. Womer. Scheduling with time dependent processing times : Review and extensions. *The Journal of the Operational Research Society*, 50(7) :711–720, 1999. 3.1.1, 3.1.1

M. R. Amin-Naseri and M. A. Beheshti-Nia. Hybrid flow shop scheduling with parallel batching. *International Journal of Production Economics*, 117(1) :185 – 196, 2009. 1.3

L. H. Andrew, A. Wierman, and A. Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Perform. Eval. Rev.*, 37(2) :39–41, October 2009. 4.2.1

E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Euro-Par 2012 Parallel Processing*, pages 128–140, Berlin, Heidelberg, 2012. Springer. 4.2.2

A. Antoniadis and C-C. Huang. Non-preemptive speed scaling. *J. of Scheduling*, 16(4) :385–394, August 2013. 4.2.1

A. Antoniadis, N. Barcelo, M. Consuegra, P. Kling, M. Nugent, K. Pruhs, and M. Scquizzato. Efficient computation of optimal energy and fractional weighted flow trade-off schedules. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science*, pages 63–74, 2014. 4.2.1

A. Antoniadis, C-C. Huang, and S. Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1102–1113, 2015. 4.2.2

J-P Arnaout, R. Musa, and G. Rabadi. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines-part ii : enhancements and experimentations. *Journal of Intelligent Manufacturing*, 2014. 1.12.1, 1.1

L. Atkins, G. Aupy, D. Cole, and K. Pruhs. Speed scaling to manage temperature. In *Theory and Practice of Algorithms in (Computer) Systems - First International (ICST) Conference, (TAPAS) 2011, Rome, Italy, April 18-20, 2011. Proceedings*, pages 9–20, 2011. 4.2.1

J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. *SIAM J. Comput.*, 37(5) : 1499–1516, 2008. 4.2.1

P. Aurich, A. Nahhas, T. Reggelin, and J. Tolujew. Simulation-based optimization for solving a hybrid flow shop scheduling problem. In *2016 Winter Simulation Conference (WSC)*, pages 2809–2819, 2016. 1.3

M. Azizoglu, E. Cakmak, and S. Kondakci. A flexible flowshop problem with total flow time minimization. *European Journal of Operational Research*, 132(3) :528 – 538, 2001. 1.3

A. Bachman and A. Janiak. Minimizing maximum lateness under linear deterioration. *European Journal of Operational Research*, 126(3) :557–566, 2000. 3.1.1

A. Bachman, A. Janiak, and M. Y. Kovalyov. Minimizing the total weighted completion time of deteriorating jobs. *Information Processing Letters*, 81(2) :81–84, 2002. 3.1.1

K. R. Baker. *Introduction to sequencing and scheduling*. Wiley, New York, Etats-Unis, 1974. 1

A.N. Balaji and S. Porselvi. Artificial immune system algorithm and simulated annealing algorithm for scheduling batches of parts based on job availability model in a multi-cell flexible manufacturing system. *Procedia Engineering*, 97 :1524 – 1533, 2014. 1.2

E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, and I. Nemparis. From preemptive to non-preemptive speed-scaling scheduling. In *Computing and Combinatorics*, pages 134–146, Berlin, Heidelberg, 2013. Springer. 4.2.2

E. Bampis, D. Letsios, and G. Lucarelli. A note on multiprocessor speed scaling with precedence constraints. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 138–142, 2014. 4.2.2

M. Bank, S. M. T. Fatemi Ghomi, F. Jolai, and J. Benhamian. Two-machine flow shop total tardiness scheduling problem with deteriorating jobs. *Applied Mathematical Modelling*, 36(11) :5418–5426, 2012. 3.1.3

N. Bansal and K. Pruhs. Speed scaling to manage temperature. *Symposium on Theoretical Aspects of Computer Science*, 3404 :460–471, 2005. 4.2.1

N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1) :3 :1–3 :39, 2007a. 4.2.1, 4.2.2

N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 805–813, 2007b. 4.2.1

N. Bansal, H-L. Chan, T-W. Lam, and L-K. Lee. Scheduling for speed bounded processors. *Automata, Languages and Programming*, 5125 :409–420, Jul 2008. 4.2.1

N Bansal, H-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 693–701, 2009. 4.2.1

N. Bansal, D. P. Bunde, H-L. Chan, and K. Pruhs. Average rate speed scaling. *Algorithmica*, 60 (4) :877–889, Aug 2011. 1.13, 4.2.1

N. Bansal, H-L. Chan, D. Katz, and K. Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(9) :209–229, 2012. 4.2.1

P. Baptiste, M. Chrobak, and C. Durr. Polynomial-time algorithms for minimum energy scheduling. *ACM Trans. Algorithms*, 8(3), July 2012. 4.2.1, 4.2.2

A. Baskar and M. A. Xavior. A new heuristic algorithm using pascal's triangle to determine more than one sequence having optimal / near optimal make span in flow shop scheduling problems. *International Journal of Computer Applications*, 39(5) :9–15, 2012. 1.2

J. Bazewicz, J. Węglarz, K. Ecker, and G. Schmidt. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 2001. 1.6

K. Belkadi, M. Gourgand, M. Nenyettou, and A. Aribi. sequencial and parallel genetic algorithms with migration for the hybrid flow shop scheduling problem. *Applied Sciences*, 6 :775–778, 2006. 1.3

A. Vignierand J.C. Billaut and C. Proust. Hybrid flowshop scheduling problems : state of the art. *Rairo- Recherche Operationnelle - Operations Research*, 33(2) :117–83, 1999. 1.13

B. D. Bingham and M. R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, pages 153–161, Dec 2008. 4.2.2

K. Brockmann, W. Dangelmaier, and N. Holthofer. *Parallel Branch & Bound Algorithm for Makespan Optimal Scheduling in Flow Shops With Multiple Processors*, pages 428–433. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. 1.3

S. Browne and U. Yechiali. Scheduling deteriorating jobs on a single processor. *Oper. Res.*, 38(3) : 495–498, 1990. 3.1, 5.2, A.1.3

D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 190–196, 2006. 4.2.1, 4.2.2

J. Y. Cai, P. Cai, and Y. Zhu. On a scheduling problem of time deteriorating jobs. *Journal of complexity*, 14(2) :190–209, 1998. 3.1.1

J. Carlier and E. Neron. An exact method for solving the multi-processor flow-shop. *RAIRO - Operations Research*, 34(1) :1 – 25, 2000. 1.3

A. Cataldo, A. Perizzato, and R. Scattolini. Production scheduling of parallel machines with model predictive control. *Control Engineering Practice*, 42 :28 – 40, 2015. ISSN 0967-0661. 4.2.2

U. K. Chakraborty and D. Laha. An improved heuristic for permutation flowshop scheduling. *Int. J. Inf. Commun. Techol.*, 1(1) :89–97, apr 2007. 1.2

H.-L. Chan, J. W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Optimizing throughput and energy in online deadline scheduling. *ACM Trans. Algorithms*, 6(1), December 2009. 4.2.1

H-L. Chan, S-H. Chan, L. Tak-Wah, L-K. Lee, R. Li, and C.-M. Liu. Competitive online algorithms for multiple-machine power management and weighted flow time. In *Theory of Computing 2013*, volume 141 of *CRPIT*, pages 11–20, 2013a. 4.2.2

S.-H. Chan, T.-W. Lam, and L.-K. Lee. Non-clairvoyant speed scaling for weighted flow time. In *Proceedings of the 18th Annual European Conference on Algorithms : Part I*, ESA'10, pages 23–35, 2010. 4.2.1

S.-H. Chan, T.-W. Lam, L.-K. Lee, C.-M. Liu, and H.-F. Ting. Sleep management on multiple machines for energy and flow time. In *Automata, Languages and Programming*, pages 219–231, Berlin, Heidelberg, 2011. Springer. 4.2.2

S.-Hang Chan, T.-W. Lam, L.-K. Lee, and J. Zhu. Nonclairvoyant sleep management and flow-time scheduling on multiple processors. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 261–270, 2013b. 4.2.2

J. Chang, W. Yan, and H. Shao. Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. In *Proceedings of the 2004 American Control Conference*, volume 2, pages 1412–1416, 2004. 1.3

P.C. Chang and S.H. Chen. Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, 11(1) :1263 – 1274, 2011. 1.1

C. Charalambous and K. Fleszar. Variable neighborhood descent for the unrelated parallel machine scheduling problem. *International Journal on Artificial Intelligence Tools*, 21(04), 2012. 1.1

I. A. Chaudhry, R. Ahmed, and A.-M. Khan. Genetic algorithm to minimize flowtime in a no-wait flowshop scheduling problem. *IOP Conference Series : Materials Science and Engineering*, 65(1), 2014. 1.2

A. Che, S. Zhang, and X. Wu. Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs. *Journal of Cleaner Production*, 156 :688 – 697, 2017. 4.2.2

J.F. Chen. Unrelated parallel machine scheduling with secondary resource constraints. *The International Journal of Advanced Manufacturing Technology*, pages 285–292, 2005. 1.1

L. Chen, N. Bostel, P. Dejax, J. Cai, and L. Xi. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research*, 181(1) :40 – 58, 2007. 1.3

L. Chen, W. Luo, and G. Zhang. Approximation algorithms for unrelated machine scheduling with an energy budget. *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, 6681 :244–254, 2011. 4.2.2

Z. L. Chen. Parallel machine scheduling with time dependent processing times. *Discrete Applied Mathematics*, 70(1) :81–93, 1996. 3.1.2

J. Cheng, F. Chu, and M. Zhou. An improved model for parallel machine scheduling under time-of-use electricity price. *IEEE Transactions on Automation Science and Engineering*, PP(99) :1–4, 2017. 4.2.2

M. Cheng, P. R. Tadikamalla, J. Shang, and B. Zhang. Two-machine flow shop scheduling with deteriorating jobs : Minimizing the weighted sum of makespan and total completion time. *Journal of the operational research society*, 66(5) :709–719, 2015. 3.1.3

T. Cheng and Q. Ding. Single machine scheduling with deadlines and increasing rates of processing times. *Acta Informatica*, 36(9) :673–692, 2000. 3.1.1

T.C.E. Cheng and Q. Ding. The complexity of single machine scheduling with two distinct deadlines and identical decreasing rates of processing times. *Computers and Mathematics with Applications*, 35(12) :95–100, 1998a. 3.1.1

T.C.E. Cheng and Q. Ding. The complexity of scheduling starting time dependent tasks with release times. *Information Processing Letters*, 65(2) :75–79, 1998b. 3.1.1

T.C.E. Cheng, Q. Ding, M.Y. Kovalyov, A. Bachman, and A. Janiak. Scheduling jobs with piecewise linear decreasing processing times. *Naval research logistics*, 50(6) :531–554, 2003. 3.1.1, 3.1.2

T.C.E. Cheng, Q. Ding, and B.M.T Lin. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1) :1–13, 2004. 3.1.1, 3.1.1

M. Dai, D. Tang, A. Giret, M. A. Salido, and W.D. Li. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, 29(5) :418 – 429, 2013. 4.2.4

M. Dai, D. Tang, Y. Xu, and W. D. Li. Energy-aware integrated process planning and scheduling for job shops. *Proceedings of the Institution of Mechanical Engineers, Part B : Journal of Engineering Manufacture*, 229(1) :13–26, 2014. 4.2.3

P. Damodaran, D.A. Diyadawagamage, O. Ghrayeb, and M. C. Vélez-Gallego. A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines. *The International Journal of Advanced Manufacturing Technology*, 58 :1131–1140, 2012. 1.1

M. R. de Paula, M. G. Ravetti, G. R. Mateus, and P. M. Pardalos. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18(2) :101–115, 2007. 1.1

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2) :182–197, 2002. 2.3.3

M. M. Dessouky, M. I. Dessouky, and S. K. Verma. Flowshop scheduling with identical jobs and uniform parallel machines. *European Journal of Operational Research*, 109(3) :620 – 631, 1998. 1.3

J-Y. Ding, S. Song, R. Zhang, R. Chiong, and C. Wu. Parallel machine scheduling under time-of-use electricity prices : New models and optimization approaches. *IEEE Transactions on Automation Science and Engineering*, 13(2) :1138–1154, 2016. 4.2.2

H. Djellab and K. Djellab. Preemptive hybrid flowshop scheduling problem of interval orders. *European Journal of Operational Research*, 137(1) :37 – 49, 2002. 1.3

J. Dong, W. Tong, T. Luo, X. Wang, J. Hu, Y. Xu, and G. Lin. An fptas for the parallel two-stage flowshop problem. *Theoretical Computer Science*, 657 :64 – 72, 2017. 1.3

E. Ebrahimi and J. Rezaeian. Unrelated parallel machines scheduling with the effect of aging and learning under multi maintenance activities. *Manufacturing Science and Technology*, 3 :25 – 31, 2015. 1.1

O. Engin and A. Doyen. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20(6) :1083 – 1095, 2004. 1.3

D. Y. Eroglu, H. C. Ozmutlu, and S. Ozmutlu. Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research*, 52(19) :5841–5856, 2014. 1.1

J. Escamilla, M. A. Salido, A. Giret, and F. Barber. A metaheuristic technique for energy-efficiency in job shop scheduling. *Constraint Satisfaction Techniques for Planning and Scheduling*, 24, 2014. 4.2.3

A. E. Ezugwu, O. J. Adeleke, and S. Viriri. Symbiotic organisms search algorithm for the unrelated parallel machines scheduling with sequence-dependent setup times. *PLOS ONE*, 13(7) :1–23, 2018. 1.1

B. Fan, S. Li, L. Zhou, and L. Zhangi. Scheduling resumable deteriorating jobs on a single machine with non-availability constraints. *Theoretical Computer Science*, 412(4) :275–280, 2011. 3.1.1

K. Fang, N. A. Uhan, F. Zhao, and J. W. Sutherland. Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research*, 206(1) :115–145, 2013. 4.2.3

W. Fang, R. Yunqing, C. Zhang, Q. Tang, and L. Zhang. Estimation of distribution algorithm for energy-efficient scheduling in turning processes. *Sustainability*, 8(8) :762, 2016. 4.2.4

L. Fanjul-Peyro and R. Ruiz. Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers & Operations Research*, 39(7) :1745 – 1753, 2012. 1.1

E. Figielska. A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources. *Computers & Industrial Engineering*, 56(1) :142 – 151, 2009. 1.3

C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization : Formulation discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, 1993. 2.3.3

M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 141–153, 1985. 2.3.2

M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, USA, 1979. ISBN 0716710447. 1.1

S. Gawiejnowicz. Scheduling deteriorating jobs subject to job or machine availability constraints. *European Journal of Operational Research*, 180(1) :472–478, 2007. 3.1.1

S. Gawiejnowicz and L. Pankowska. Scheduling jobs with varying processing times. *Information Processing Letters*, 54(3) :175–178, 1995. 3.1.1

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13 :533–549, 1986. 2.3.1

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5 :287 – 326, 1979. 1.6, 1.13, 5.1

S. Guirchoun, P. Martineau, and J.C. Billaut. Total completion time minimization in a computer system with a server and two parallel processors. *Computers & Operations Research*, 32(3) : 599–611, 2005. 1.3

P. Guo, W. Cheng, and Y. Wang. A general variable neighborhood search for single-machine total tardiness scheduling problem with step-deteriorating jobs. *Management*, 10(4) :1071–1090, 2014. 3.1.1

A. Gupta, R. Krishnaswamy, and K. Pruhs. Scalably scheduling power-heterogeneous processors. *CoRR*, abs/1105.3748, 2011. 4.2.2

J. N. D. Gupta and S. K. Gupta. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering*, 14(4) :387–393, 1988. 3.1

J. N.D. Gupta and E. A. Tunc. Scheduling a two-stage hybrid flowshop with separable setup and removal times. *European Journal of Operational Research*, 77(3) :415 – 428, 1994. 1.3

J.N.D. Gupta, A.M.A. Hariri, and C.N. Potts. Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69(0) :171–191, 1997. 1.3

X. Han, T-W. Lam, L-K. Lee, I. K.K. To, and P. W.H. Wong. Deadline scheduling and power management for speed bounded processors. *Theoretical Computer Science*, 411(40/42) :3587 – 3600, 2010. 4.2.1

M. Haouari, L. Hidri, and A. Gharbi. Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64(1) :107–124, 2006. 1.3

A. M. A. Hariri and C. N. Potts. Heuristics for scheduling unrelated parallel machines. *Comput. Oper. Res.*, 18(3) :323–331, 1991. 1.1

L. Hidri and M. Haouari. Bounding strategies for the hybrid flow shop scheduling problem. *Applied Mathematics and Computation*, 217(21) :8248 – 8263, 2011. 1.3

L. Hidri, A. Gharbi, and M. A. Louly. Efficient bounding schemes for the two-center hybrid flow shop scheduling problem with removal times, 2014. 1.3

H.L., Chan S.H. Chan, T.W. Lam, L.K. Lee, R. Li, and C.M. Liu. Competitive online algorithms for multiple-machine power management and weighted flow time. In *Theory of Computing 2013 (CATS 2013)*, volume 141 of *CRPIT*, pages 11–20, 2013. 4.2.2

K. I-J. Ho, J. Y-T. Leung, and W-D. Wei. Complexity of scheduling tasks with time-dependent execution times. *Information Processing Letters*, 48(6) :315–320, 1993. 3.1.1

J. H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984. 2.3.1

Y. S. Hsieh and D. L. Bricker. Scheduling linearly deteriorating jobs on multiple machines. *Computers and Industrial Engineering*, 32(4) :727–734, 1997. 3.1.2

C. J. Hsu, M. Ji, J. Y. Guo, and D. L. Yang. Unrelated parallel-machine scheduling problems with aging effects and deteriorating maintenance activity. *Information Sciences*, 253 :163–169, 2013. 3.1.2, 3.1.2

L. C. Hsu. Some identities involving three kinds of counting numbers. *ArXiv*, 2009. 5.2

S. Hu, F. Liu, Y. He, and T. Hu. An on-line approach for energy efficiency monitoring of machine tools. *Journal of Cleaner Production*, 27 :133 – 140, 2012. 4.2

C.-C. Huang and S. Ott. New results for non-preemptive speed scaling. In *Mathematical Foundations of Computer Science 2014*, pages 360–371, Berlin, Heidelberg, 2014. Springer. 4.2.1

W. Huang and S. Li. A two-stage hybrid flowshop with uniform machines and setup times. *Mathematical and Computer Modelling*, 27(2) :27 – 45, 1998. 1.3

E. Ignall and L. Schrage. Application of the branch and bound technique to some flow-shop scheduling problems. *Oper. Res.*, 13(3) :400–412, June 1965. 1.2

IEA International Energy Agency. Key world energy statistics 2016, 2016. (document), 4.1, 4.2, 4.1

S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4), November 2007. 4.2.2

A. Jafari, H. Khademi Zare, M.M. Lotfi, and R. Tavakkoli-Moghaddam. A note on "minimizing makespan in three machine flow shop with deteriorating jobs". *Computers & Operations Research*, 72 :93–96, 2016. 3.1.3

M. Ji and T.C.E. Cheng. An fptas for scheduling jobs with piecewise linear decreasing processing times to minimize makespan. *Information Processing Letters*, 102(2) :41–47, 2007. 3.1.1, 3.1.2

M. Ji and T.C.E. Cheng. Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science*, 410(38) :3761–3768, 2009. 3.1.2

M. Ji, Y. He, and T.C.E. Cheng. Scheduling linear deteriorating jobs with an availability constraint on a single machine. *Theoretical Computer Science*, 362(1) :115–126, 2006. 3.1.1

M. Ji, X. Tang, X. Zhang, and T.C. E. Cheng. Machine scheduling with deteriorating jobs and dejong's learning effect. *Computers and Industrial Engineering*, 91 :42–47, 2016. 3.1.2

Z. Jiang, L. Zuo, and E. Mingcheng. Study on multi-objective flexible job-shop scheduling problem considering energy consumption. *Journal of Industrial Engineering and Management*, 7(3) :589–604, 2014. 4.2.4

Z. Jin, Z. Yang, and T. Ito. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100(2) :322 – 334, 2006. 1.3

C. Kahraman, O. Engin, A. Kaya, and M. K. Yilmaz. An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. *International Journal of Computational Intelligence Systems*, 1(2) :134–147, 2008. 1.3

G. Kant and K. Sigh Sangwan. Predictive modeling for power consumption in machining using artificial intelligence techniques. *Procedia CIRP*, 26 :403 – 407, 2015. 4.2

F. Keller, C. Schonborn, and G. Reinhart. Energy-orientated machine scheduling for hybrid flow shops. *Procedia CIRP*, 29 :156 – 161, 2015. 4.2.4

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598) :671–680, 1983. 2.3.1

J. Koo and Bl. Kim. Some comments on "optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency". *The International Journal od Advanced Manufacturing Technology*, 86 :2803–2806, 2016. 4.2.2

M. Kovalyov and W. Kubiak. A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. *Journal of heuristics*, 3 :287–297, 1998. 3.1.1

W. Kubiak and S. van de Velde. Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics : an International Journal*, 45(5) :511–523, 1998. 3.1.1

G. Kumar and S. Shannigrahi. New online algorithm for dynamic speed scaling with sleep state. *Theoretical Computer Science*, 593 :79 – 87, 2015. 4.2.1

K.M. Senthil Kumar, V. Selladurai, K. Raja, and K. Elangovan. Ant colony approach for makespan minimization on unrelated parallel machines. *International Journal of Engineering Science and Technology*, 3, 2011. 1.1

A. S. Kunnathur and S. K. Gupta. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47(1) :56–64, 1990. 3.1.1

W. Kuo and D. L. Yang. Parallel-machine scheduling with time dependent processing times. *Theoretical Computer Science*, 393(1) :204–210, 2008. 3.1.2

M. E. Kurz and R. G. Askin. Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159(1) :66 – 82, 2004. 1.3

G. J. Kyparisis and C. Koulamas. A note on weighted completion time minimization in a flexible flow shop. *Operations Research Letters*, 29(1) :5 – 11, 2001. 1.3

G. J. Kyparisis and C. Koulamas. A note on makespan minimization in two-stage flexible flow shops with uniform machines. *European Journal of Operational Research*, 175(2) :1321 – 1327, 2006a. 1.3

G. J. Kyparisis and C. Koulamas. Flexible flow shop scheduling with uniform parallel machines. *European Journal of Operational Research*, 168(3) :985 – 997, 2006b. 1.3

T-W. Lam, L-K. Lee, I. K. K. To, and P. W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '08, pages 256–264, 2008a. 4.2.2

Tak-Wah Lam, Lap-Kei Lee, Isaac K. To, and Prudence W. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the 16th Annual European Symposium on Algorithms*, ESA '08, pages 647–659, 2008b. 4.2.1, 4.2.4

C. Y. Lee and G. L. Vairaktarakis. Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16(3) :149 – 158, 1994. 1.3

W. C. Lee and C. C. Wu. Scheduling linear deterioration jobs to minimize makespan with an availability constraint on a single machine. *Information processing letters*, 87 :89–93, 2003. 3.1.1

W. C. Lee and C. C. Wu. Multi-machine scheduling with deteriorating jobs and scheduled maintenance. *Applied mathematical modelling*, 32(3) :362–373, 2008. 3.1.2

W. C. Lee, C. C. Wu, and Y. H. Chung. Scheduling deteriorating jobs on a single machine with release times. *Computers and Industrial Engineering*, 54(3) :441–452, 2008. 3.1.1

W. C. Lee, W. C. Yeh, and Y. H. Chung. Total tardiness minimization in permutation flowshop with deterioration consideration. *Applied Mathematical Modelling*, 38(13) :3081–3092, 2014. 3.1.3

D. Lei and X. Guo. Hybrid flow shop scheduling with not-all-machines options via local search with controlled deterioration. *Computers & Operations Research*, 65 :76 – 82, 2016. 1.3

M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.*, 35(3) :658–671, September 2005. 4.2.1, 4.2.2

M. Li, A. C. Yao, and F. F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *Proceedings of the National Academy of Sciences of the United States of America*, 103(11) :3983–3987, 2006. 4.2.2

M. Li, F. F. Yao, and H. Huan. An o(n^2) algorithm for computing optimal continuous voltage schedules. ArXiv, 2014. 4.2.1, A.1.2

S. Li and J. Yuan. Parallel-machine scheduling with deteriorating jobs and rejection. *Theoretical Computer Science*, 411(40) :3642–3650, 2010. 3.1.2

Y. Li, G. Li, L. Sun, and Z. Xu. Single machine scheduling of deteriorating jobs to minimize total absolute differences in completion times. *International Journal of Production Economics*, 118(2) : 424–429, 2009. 3.1.1

Z. Li, H. Yang, S. Zhang, and G. Liu. Unrelated parallel machine scheduling problem with energy and tardiness cost. *The International Journal of Advanced Manufacturing Technology*, 2016. 4.2.2

S.-W. Lin and K.-C. Ying. A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems. *International Journal of Production Research*, 53 (4) :1065–1076, 2015. 6.2.2, A.2.3

C.H. Liu and D.H. Huang. Reduction of power consumption and carbon footprints by applying multi-objective optimisation via genetic algorithms. *International Journal of Production Research*, 52(2) :337–352, 2014. 4.2.1, 4.2.4

Y. Liu, H. Dong, N. Lohsr, S. Petrovic, and N. Gindy. An investigation into minimising total energy consumption and total weighted tardiness in job shops. *Journal of Cleaner Production*, 65 :87–96, 2014. 4.2.3

C. Low, C.-J. Hsu, and C.-T. Su. A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. *Computers & Operations Research*, 35(3) :845 – 853, 2008. 1.3

H. Luo, B. Du, G. Q. Huang, H. Chen, and X. Li. Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics*, 146(2) :423 – 439, 2013. 4.2.4

L. Mahdavi, V. Zarezadeh, and P. Shahnazari-Shahrezaei. Flexible flowshop scheduling with equal number of unrelated parallel machines. *Journal of Industrial Engineering, International*, 7(13) : 74–83, 2011. 1.3

W. H. Marlow. *Mathematics for operations research*. Dover publications, New York, United states, 1993. 2

O. Masmoudi, A. Yalaoui, Y. Ouazene, and H. Chehade. Lot-sizing in flow-shop with energy consideration for sustainable manufacturing systems. *IFAC-Papers OnLine*, 48(3) :727–732, 2015. 4.2.3

G. May, B. Stahl, M. Taisch, and V. Prahbu. Multi-objective genetic algorithm for energy-efficient job shop scheduling. *International Journal of Production Research*, 53(23) :7071–7089, 2015. 4.2.2, 4.2.3

M. M. Mazdeh, F. Zaerpour, A. Zareei, and A. Hajinezhad. Parallel machines scheduling to minimize job tardiness and machine deteriorating cost with deteriorating jobs. *Applied Mathematical Modelling*, 34(6) :1498–1510, 2010. 5.1, A.1.2

N. Megow and J. Verschae. Scheduling on a machine with varying speed : Minimizing cost and energy via dual schedules. *CoRR*, abs/1211.6216, 2012. 4.2.1

O. Mel'nikov and Y. Shafransky. Parametric problem in scheduling theory. *Cybernetics*, 15 :352–357, 1980. 3.1.1

V. Modrak and R. S. Pandian. Flowshop scheduling algorithm to minimize completion time for n jobs m machines problem. *Techniki Vjesnik*, 17(3) :273–278, 2010. 1.2

H. Mokhtari and A. Hasani. An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. *Computers Chemical Engineering*, 104 :339–352, 2017. 4.2.4

J.Y. Moon, K. Shin, and J. Park. Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency. *The International Journal of Advanced Manufacturing Technology*, 2013. 4.2.2

T. Morton and D. W. Pentico. *Heuristic scheduling systems*. Wiley, New York, Etats-Unis, 1993. 1.1

G. Mosheiov. V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39(6) : 979–991, 1991. 3.1.1

G. Mosheiov. Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, 21(6) :653–659, 1994. 3.1.1

G. Mosheiov. Scheduling jobs with step-deterioration ; minimizing makespan on a single and multi-machine. *Computers and industrial ingineering*, 28(4) :869–879, 1995. 3.1.1, 3.1.2

G. Mosheiov. Multi-machine scheduling with linear deterioration. *INFOR : Information Systems and Operational Research*, 36(4) :205–214, 1998. 3.1.2

G. Mosheiov. Complexity analysis of job-shop scheduling with deteriorating jobs. *discrete applied mathematics*, 117(1-3) :195–209, 2002. 3.1.3

G. Mosheiov. A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, 41 (8) :883–886, 2005. 3.1.1

S. M. Mousavi, M. Mousakhani, and M. Zandieh. Bi-objective hybrid flow shop scheduling : a new local search. *The International Journal of Advanced Manufacturing Technology*, 64(5) :933–950, 2013. 1.3

G. Mouzon. *Operational methods and models for minimization of energy consumption in manufacturing environment*. PhD thesis, Wichita State University, May 2008. 4.2.1, 4.2.2, 5.1

G. C. Mouzon and M. B. Yildirim. A framework to minimize total energy consumption and total tardiness on a single machine. *International Journal of Sustainable Engineering*, 1(2) :105–116, 2008. 4.2, 4.2.1

B. Naderi and H. Sadeghi. A multi-objective simulated annealing algorithm to solving flexible no-wait flowshop scheduling problems with transportation times. *Journal of Optimization in Industrial Engineering*, 5(11) :33–41, 2012. 1.3

K. Nailwal, D. Gupta, and K. Jeet. Heuristics for no-wait flow shop scheduling problem. *International Journal of Industrial Engineering Computations*, 7(4) :671–680, 2016. 1.2

E. G. Negenman. Local search algorithms for the multiprocessor flow shop scheduling problem. *European Journal of Operational Research*, 128(1) :147 – 158, 2001. 1.3

C.T. Ng, T.C.E. Cheng, A. Bachman, and A. Janiak. Three scheduling problems with deteriorating jobs to minimize the total completion time. *Information Processing Letters*, 81(6) :327–333, 2002. 3.1.1

C.T. Ng, J.B. Wang, T.C.E. Cheng, and L.L. Liu. A branch-and-bound algorithm for solving a two-machine flow shop problem with deteriorating jobs. *Computers & Operations Research*, 37 (1) :83–90, 2010. 3.1.3

G. Nicolo, M.A. Salido, S. Ferrer, A.Giret, and F. Barber. A multi-agent approach using dynamic constraints to solve energy-aware unrelated parallel machine scheduling problem with energy-dependent and sequence-dependent setup time. In *COPLAS'2017*, pages 31–37, 2017. 4.2.2

M. Nikabadi and R. Naderi. A hybrid algorithm for unrelated parallel machines scheduling. *International Journal of Industrial Engineering Computations*, 7(4) :681–702, 2016. 1.1

E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1) :160 – 175, 1996. 1.2

C. Oguz, B. M.T. Lin, and T.C. E. Cheng. Two-stage flowshop scheduling with a common second-stage machine. *Computers & Operations Research*, 24(12) :1169 – 1174, 1997. 1.3

D. Oron. Single machine scheduling with simple linear deterioration to minimize total absolute deviation of completion times. *Computers & Operations Research*, 35(6) :2071–2078, 2008. 3.1.1

G. M. G. Pachedo, L. E. R. Polo, and V. P. M. Nino. Flexible flowshop problem minimizing total flow time and makespan using evolutionary algorithm. In *LACCEI'2013*, pages 1–9, 2013. 1.3

Q.-K. Pan and Y. Dong. An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. *Information Sciences*, 277 :643 – 655, 2014. 1.3

Q.-K. Pan, L. Wang, J.-Q. Li, and J.-H. Duan. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega*, 45 :42 – 56, 2014. 1.3

C. D. Paternina-Arboleda, J. R. Montoya-Torres, M. J. Acero-Dominguez, and M. C. Herrera-Hernandez. Scheduling jobs on a k-stage flexible flow-shop. *Annals of Operations Research*, 164 (1) :29–40, 2008. 1.3

A. Pechmann and I. Schöler. *Optimizing Energy Costs by Intelligent Production Scheduling*, pages 293–298. Springer, Berlin, Heidelberg, 2011. 4.2

X. Piao and M. Park. On-line dynamic voltage scaling for edzl scheduling on symmetric multiprocessor real-time systems. *nternational Journal of Multimedia and Ubiquitous Engineering*, 10(7) : 171–182, 2015. 4.2.2

M.-C. Portmann, A. Vignier, D. Dardilhac, and D. Dezalay. Branch and bound crossed with ga to solve hybrid flowshops. *European Journal of Operational Research*, 107(2) :389 – 400, 1998. 1.3

K. Pruhs, R. Van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *Proceedings of the Third International Conference on Approximation and Online Algorithms*, WAOA'05, pages 307–319, 2006. 4.2.1, 4.2.2

G. Rabadi, R. J. Moraga, and A. Al-Salem. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1) :85–97, 2006. 1.12.1

M. Rabiee, R. Sadeghi Rad, M. Mazinani, and R. Shafaei. An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 71(5) :1229–1245, 2014. 1.3

C. R. Ren and L. Y. Kang. An approximation algorithm for parallel machine scheduling with simple linear deterioration. *Journal of Shanghai University*, 11(4) :351–354, 2007. 3.1.2

F. Riane, A. Artiba, and S. E. Elmaghraby. A hybrid three-stage flowshop problem : Efficient heuristics to minimize makespan. *European Journal of Operational Research*, 109(2) :321 – 329, 1998. 1.3

F. Riane, A. Artiba, and S. E. Elmaghraby. Sequencing a hybrid two-stage flowshop with dedicated machines. *International Journal of Production Research*, 40(17) :4353–4380, 2002. 1.3

S. U. Sapkal and D. Laha. A heuristic for no-wait flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 68(5) :1327–1338, 2013. 1.2

T. J. Sawik. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 31(13) :39 – 52, 2000. 1.3

Y. Seow, N. Goffin, S. Rahimifard, and E. Woolley. A design for energy minimization approach to reduce energy consumption during the manufacturing phase. *Energy*, 109 :894 – 905, 2016. 4.2

S.V. Sevastianov. Geometrical heuristics for multiprocessor flowshop scheduling with uniform machines at each stage. *Journal of Scheduling*, 5(3), 2002. 1.3

E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2) :127 – 133, 2005. 1.1

W. Shi, X. Song, and J. Sun. Automatic heuristic generation with scatter programming to solve the hybrid flow shop problem. *Advances in Mechanical Engineering*, 7(2) :587038, 2015. 1.3

D.-F. Shiau, S.-C. Cheng, and Y.-M. Huang. Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm. *Expert Systems with Applications*, 34(2) :1133 – 1143, 2008. 1.3

P. Siarry. *Métaheuristiques*. eyrolles, 2014. 2

H. Soewandi and S. E. Elmaghraby. Sequencing on two-stage hybrid flowshops with uniform machines to minimize makespan. *IIE Transactions*, 35(5) :467–477, 2003. 1.3

T. Stock and G. Seliger. Multi-objective shop floor scheduling using monitored energy data. *Procedia CIRP*, 26 :510–515, 2015. 4.2.4

T. Stutzle. An ant approach to the flow shop problem. In *In proceedings of the 6th European congress of intelligent techniques & soft computing (EUFIT'98)*, pages 1560–1564, 1997. 1.2

P.S. Sundararaghavan and A.S. Kunnathur. Single machine scheduling with start time dependent processing times : Some solvable cases. *European journal of operational research*, 78(3) :394–403, 1994. 3.1.1

V. Suresh. A note on scheduling of two-stage flow shop with multiple processors. *International Journal of Production Economics*, 49(1) :77 – 82, 1997. 1.3

V. S. Tanaev, V. S. Gordon, and Y. M. Shafransky. *Scheduling Theory. Single-Stage Systems*, volume 284. Springer Netherlands, 1994. 3.1

M. Tigane, M. Dahane, and M. Boudhar. Multiobjective approach for deteriorating jobs scheduling for a sustainable manufacturing system. *The International Journal of Advanced Manufacturing Technology*, 101 :1939–1957, 2019. 5, A.1

C.-T. Tseng, C.-J. Liao, and T.-X. Liao. A note on two-stage hybrid flowshop scheduling with missing operations. *Computers & Industrial Engineering*, 54(3) :695 – 704, 2008. 1.3

L.-Y. Tseng and Y.-T. Lin. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1) :84 – 92, 2009. 1.2

S. Verma and M. Dessouky. Multistage hybrid flowshop scheduling with identical jobs and uniform parallel machines. *Journal of Scheduling*, 2(3) :135–150, 1999. 1.3

W. W. Li, A. Zein, S. Kara, and C. Herrmann. An investigation into fixed energy consumption of machine tools. In J. Hesselbach and C. Herrmann, editors, *Glocalized Solutions for Sustainability in Manufacturing*, pages 268–273, Berlin, Heidelberg, 2011. Springer. 4.2

C. Wang, X. Li, and Q. Wang. Accelerated tabu search for no-wait flowshop scheduling problem with maximum lateness criterion. *European Journal of Operational Research*, 206(1) :64 – 72, 2010a. 1.2

J B. Wang. Single machine scheduling with decreasing linear deterioration under precedence constraints. *Computers and mathematics with applications*, 58(1) :95–103, 2009a. 3.1.1

J. B. Wang and M. Z. Wang. Minimizing makespan in three machine flow shops with deteriorating jobs. *Computers & operations research*, 40(2) :547–557, 2013. 3.1.3

J. B. Wang and Z. Q. Xia. Scheduling jobs under decreasing linear deterioration. *Information Processing Letters*, 94(2) :63–69, 2005. 3.1.1

J. B Wang and Z. Q. Xia. Flow shop scheduling with deteriorating jobs under dominating machines. *Omega*, 34(4) :327–336, 2006. 3.1.3

J. B. Wang, C.T. Daniel Ng, T.C.E. Cheng, and L. L. Liu. Minimizing total completion time in a two-machine flow shop with deteriorating jobs. *Applied Mathematics and Computation*, 180(1) : 185–193, 2006. 3.1.3

J. B. Wang, C.T. Ng, and T.C.E. Cheng. Single-machine scheduling with deteriorating jobs under a series parallel graph constraint. *Computers & Operations Research*, 35(8) :2684–2693, 2008. 3.1.1

J. B. Wang, X. Huang, X. Y. Wang, N. Yin, and L. Y. Wang. Learning effect and deteriorating jobs in the single machine scheduling problems. *Applied Mathematical Modelling*, 33(10) :3848–3853, 2009. 3.1.1

J.B. Wang. Single machine scheduling with learning effects and deteriorating jobs. *Computers and Industrial Engineering*, 57(4) :1452–1456, 2009b. 3.1.1

J.B. Wang, J.J. Wang, and P. Ji. Scheduling jobs with chain precedence constraints and deteriorating jobs. *Journal of the operational research society*, 62(9) :1767–1770, 2011a. 3.1.1

L. Wang, Q.-K. Pan, and M. F. Tasgetiren. Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications*, 37(12) : 7929 – 7936, 2010b. 1.2

S. Wang, X. Wang, J. Yu, S. Ma, and M. Liu. Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production*, 193 :424–440, 2018. 4.2.2

X. Wang and T.C.E. Cheng. Single-machine scheduling with deteriorating jobs and learning effects to minimize the makespan. *European Journal of Operational Research*, 178(1) :57–70, 2007. 3.1.1

Y. Y. Wang, M. Z. Wang, and J. B. Wang. Flow shop scheduling to minimize makespan with decreasing time-dependent job processing times. *Computers and Industrial Engineering*, 60(4) : 840–844, 2011b. 3.1.3

Z. Wang, W. Xing, and F. Bai. No-wait flexible flowshop scheduling with no-idle machines. *Operations Research Letters*, 33(6) :609 – 614, 2005. 1.3

B. Wardono and Y. Fathi. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. *European Journal of Operational Research*, 155(2) :380 – 401, 2004. 1.3

G. J. Woeginger. Scheduling with time-dependent execution times. *Information processing letters*, 54(3) :155–156, 1995. 3.1.1

Y. Wu, M. Wang, and J. Wang. Some single-machine scheduling with both learning and deterioration effects. *Applied Mathematical Modelling*, 35(8) :3731–3736, 2011. 3.1.1

J. Xie and X. Wang. Complexity and algorithms for two-stage flexible flowshop scheduling with availability constraints. *Computers and Mathematics with Applications*, 50(10) :1629 – 1638, 2005. 1.3

D. L. Yang and W. H. Kuo. Scheduling with deteriorating jobs and learning effects. *Applied Mathematics and Computation*, 218(5) :2069–2073, 2011. 3.1.1

L. Yang-Kuei and L. Chi-Wei. Dispatching rules for unrelated parallel machine scheduling with release dates. *The International Journal of Advanced Manufacturing Technology*, 67 :269–279, 2013. 1.1

F. Yao, Demers A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–382, 1995. 4.2.1

V. Yaurima, L. Burtseva, and A. Tchernykh. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Computers & Industrial Engineering*, 56(4) :1452 – 1463, 2009. 1.3

M. Yazdani and B. Naderi. Modeling and scheduling no-idle hybrid flow shop problems. *Journal of Optimization in Industrial Engineering*, 10(21) :59–66, 2016. 1.3

H. Ye, W. Li, A. Abedini, and B. Nault. An effective and efficient heuristic for no-wait flow shop production to minimize total completion time. *Computers & Industrial Engineering*, 108 :57 – 69, 2017. 1.2

W. Yi, L. Gao, Y. Zhou, and X. Li. Differential evolution algorithm with variable neighborhood search for hybrid flow shop scheduling problem. In *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 233–238, May 2016. 1.3

K. P. Yoon and C-L. Hwang. *Lecture notes in Economics and Mathematical systems*. Springer-Verlag, 1981. 5.5

M. Zandieh and E. Rashidi. An effective hybrid genetic algorithm for hybrid flow shops with sequence dependent setup times and processor blocking. *Journal of Industrial Engineering*, 4 :51– 58, 2009. 1.2

M. Zandieh, S.M.T. Fatemi Ghomi, and S.M. Moattar Husseini. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 180(1) :111 – 127, 2006. 1.3

X. Zhang, D. Xu, D. Du, and C. Miao. Approximate algorithms for unrelated machine scheduling to minimize makespan. *Journal of Industrial and Management Optimization*, 12 :771–779, 2016. 1.1

C. Zhao and H. Tang. Two machine flow shop scheduling with deteriorating jobs and chain precedence constraints. *International journal of production economics*, 136(1) :131–136, 2012. 3.1.3

E. Zitzler, M. Laumanns, and L. Thiele. Spea2 : Improving the strength pareto evolutionary algorithm. *TIK-Report*, 103, 07 2001. 2.3.3