N° d'ordre : 02/2012-D/MT

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

THÈSE

Présentée pour l'obtention du grade de **DOCTEUR**

En **Mathématiques**
Spécialité : **Recherche Opérationnelle**

Par

HANED Amina

Thème :

ORDONNANCEMENT SOUS CONTRAINTES ADDITIONNELLES : PREEMPTION ET TRANSPORT

Soutenue publiquement, le 05/04/2012, devant le jury composé de :

M. BERRACHEDI Abdelhafid, Professeur, USTHB	Président
M. BOUDHAR Mourad, Professeur, USTHB	Directeur de thèse
M. BILLAUT Jean-Charles, Professeur, Université de Tours, France	Co-Directeur de thèse
Mlle BOUCHEMAKH Isma, Professeur, USTHB	Examinatrice
M. DERBALA Ali, Maître de Conférences /A, USD Blida	Examinateur
M. OUAFI Rachid, Professeur, USTHB	Examinateur

Remerciements

Je remercie Allah qui m'a donné la foi, la santé et le courage d'amener à terme ce travail de recherche et d'achever cette thèse.

Du fond de mon cœur je remercie mes chers parents qui illuminent et guident mes pas par leur amour. Mes parents qui m'ont soutenue et encouragée, qui sont à mes côtés et qui font de leur mieux pour mon bien-être.

Toute ma gratitude et mes plus grands remerciements vont à Monsieur Mourad BOUDHAR, mon directeur de thèse, Professeur à l'U.S.T.H.B., de m'avoir proposé ce thème et de m'avoir intégré dans son équipe de recherche. Je le remercie de m'avoir fait confiance, de croire en moi et de me faire profiter de son expérience et de son savoir-faire. Je le remercie également de m'avoir guidée dans mes recherches, pour tous les conseils et toutes les remarques pertinentes qui ont été très précieux et indispensables pour la réalisation de ce travail. Je le prie de croire à ma sincère reconnaissance et ma respectueuse estime pour son soutient et aide, pour sa présence et patience durant toutes ces années.

Mes remerciements vont aussi à Monsieur Jean-Charles BILLAUT, co-directeur de thèse, Professeur à l'université de Tours (France), d'avoir accepté de codiriger cette thèse. Je le remercie pour les invitations et l'accueil au laboratoire d'informatique (LI) de l'université de Tours, durant mes séjours scientifiques qui ont été très bénéfiques pour l'avancement et l'achèvement de ce travail.

J'exprime ma plus grande reconnaissance à Monsieur Ameur SOUKHAL, Maître de Conférences à l'université de Tours. Je le remercie d'avoir accepté de co-encadrer cette thèse et d'avoir suivi sa progression de loin. Je le remercie aussi pour son accueil au sein du laboratoire d'informatique (LI) de l'université de Tours. Je lui exprime mes plus grands remerciements pour tout le temps qu'il m'a consacré malgré son emploi du temps chargé, pour son aide, pour toutes ses remarques et conseils qui ont contribué à la réalisation de ce travail. Je le remercie également pour sa patience.

Je remercie tous les membres du jury :
Monsieur BERRACHEDI Abdelhafid, Professeur à l'U.S.T.H.B., de m'avoir fait l'honneur de présider ce jury de thèse.
Mademoiselle BOUCHEMAKH Isma, Professeur à l'U.S.T.H.B.,
Monsieur DERBALA Ali, Maître de Conférences à l'université de Blida (U.S.D.B),
Monsieur OUAFI Rachid, Professeur à l'U.S.T.H.B,
D'avoir accepté de participer à ce jury et d'évaluer ce travail.

Mes remerciements vont également à tous mes professeurs de l'U.S.T.H.B.

Je reconnais que mes sœurs Karima et Meriem ont vraiment été patientes surtout ces derniers mois, je les remercie d'être à mes côtés. Un remerciement particulier à mes cousines Chafika, Amel, Lamia et Zoubida qui ont partagé des moments de pression avec moi.

Je remercie tous les membres de ma famille, spécialement mes tantes Aicha et Chafika pour le temps qu'elles m'ont accordé et je n'oublie pas Atika, Naima et Hamida ainsi que mon oncle Hakim.

Je présente mes remerciements à mes ami(e)s et collègues pour leurs encouragements et soutien. Je cite particulièrement Amine, Nadjat, Nesrine, Wafaa, … la liste est longue !

Je saisis aussi l'occasion pour remercier tous les membres de l'équipe de recherche en ordonnancement de l'U.S.T.H.B pour leur appui et ceux du laboratoire LI de Tours pour leur agréable accueil.

Du fond de mon cœur MERCI.

Résumé :

Le travail présenté dans cette thèse, traite le problème d'ordonnancement de tâches indépendantes sur des machines parallèles identiques. L'objectif consiste à minimiser la date de fin de traitement (*makespan*). La préemption des tâches est autorisée, de plus un temps additionnel correspondant aux délais de transport des tâches préemptées est pris en considération.

Au début de cette thèse, nous avons donné quelques rappels sur la complexité des algorithmes et la complexité des problèmes d'optimisation combinatoire ainsi que les notions de base de la théorie de l'ordonnancement. Nous avons exposé par la suite les différentes méthodes de résolution. Après la définition du problème et les motivations qui nous ont incités à cette étude, un état de l'art concernant les problèmes d'ordonnancement sur machines parallèles avec plusieurs contraintes est présenté. Puis, nous nous sommes consacrés à l'étude du problème posé. Nous avons montré que c'est un problème NP-difficile même si les durées opératoires des tâches sont identiques et les délais de transport sont constants. Nous avons traité certains sous problèmes polynomiaux pour lesquels des algorithmes de résolution sont proposés. Une méthode exacte de type programmation dynamique ainsi qu'un schéma d'approximation complètement polynomial ($FPTAS$) sont proposés pour la résolution du problème à deux machines. La résolution du problème général se fait à l'aide d'une méthode en deux phases. Cette méthode consiste à déterminer la meilleure séquence des machines, dans la première phase, par l'utilisation d'une métaheuristique de type algorithme de colonies de fourmis. L'affectation des tâches, qui constitue la deuxième phase de la résolution, est faite par des heuristiques basées sur le concept des algorithmes de liste. Enfin, des expérimentations numériques pour évaluer les méthodes proposées sont établies.

Mots clés :

Ordonnancement, machines parallèles, préemption, délais de transport, complexité, programmation dynamique, $FPTAS$, heuristiques, métaheuristiques.

Abstract :

The work presented in this thesis deals with the problem of scheduling independent jobs on identical parallel machines. The objective is to minimize the total completion time (makespan). Jobs preemption is allowed and an additional time corresponding to transportation delays of the preempted jobs is considered.

At the beginning, we recall the concept of algorithms and combinatorial optimization problems complexity. We give the basics definitions of the scheduling theory. Also, we have presented the different resolution methods. After defining the problem and motivations that led us to this study, a state-of-the-art concerning the scheduling problems on parallel machines with several constraints is presented. Then, we have shown that the problem is NP-hard even if the processing time of jobs are identical and the transportation delays are constant. We treated some sub-problems, polynomial algorithms are proposed to solve them. A dynamic programming algorithm and a fully polynomial approximation scheme ($FPTAS$) are proposed to solve the problem with two machines. The resolution of the general problem is given by a method in two phases. This method determines the best sequence of machines, in the first phase, through the use of an ant colony optimization metaheuristic. The assignment of jobs, which is the second phase of the resolution, is done by heuristics based on the concept of the list algorithms. Finally, numerical experiments to evaluate the proposed methods are established.

Keywords :

Scheduling, parallel machines, preemption, transportation delays, complexity, dynamic programming, $FPTAS$, heuristics, metaheuristics.

Table des matières

Introduction générale

Cette thèse s'inscrit dans le domaine de l'ordonnancement qui constitue une branche de la recherche opérationnelle. La recherche opérationnelle est l'ensemble des méthodes scientifiques utilisables pour élaborer de meilleures décisions. La recherche opérationnelle est une science qui associe les mathématiques, l'économie et l'informatique. Elle propose des modèles conceptuels pour analyser des situations complexes et offre des méthodes pour aider les gestionnaires à prendre des décisions en se basant sur des modèles et des méthodes scientifiques adaptés. Les origines de la recherche opérationnelle remontent à la deuxième guerre mondiale, où elle a été appliquée pour répondre aux questions d'ordre militaire tel que l'implantation optimale de radars de surveillance. Depuis, son domaine d'application s'est élargi pour toucher les problèmes économiques et industriels.

Parmi les problèmes qui peuvent être traités à l'aide des techniques de la recherche opérationnelle, les problèmes combinatoires. Ces problèmes sont caractérisés par le fait qu'ils possèdent un grand nombre de solutions possibles pour un critère donné. Le but est de chercher la meilleure solution ou une bonne solution. Cette partie de la recherche opérationnelle est connue sous le nom de l'optimisation combinatoire et les problèmes d'ordonnancement en font partie.

Un problème d'ordonnancement vise à organiser au cours du temps l'exécution d'un certain nombre de tâches en utilisant un ensemble de ressources. Les problèmes d'ordonnancement trouvent des applications en industrie où il faut produire aux moindres coûts, dans ce cas on parle de l'ordonnancement des ateliers de production. Ils trouvent d'autres applications en génie civil lors du suivi des projets, en administration pour la gestion du personnel et l'emploi du temps et en informatique pour l'allocation des processeurs à l'exécution des programmes.

Comme il a été déjà mentionné, les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire. Ils intègrent les problèmes d'affectation puisqu'il s'agit d'affecter les tâches aux ressources, de plus il faut tenir compte des différentes contraintes relatives aux tâches, aux ressources et celles caractérisant le mode de traitement.

Vu la diversité de leurs applications, les problèmes d'ordonnancement font l'objet de nombreux travaux scientifiques. Les recherches ont pris deux principaux axes. D'une part l'étude de l'aspect théorique des problèmes d'ordonnancement, portant essentiellement sur la complexité et les éventuelles relations liant ces problèmes entre eux et leur relation avec d'autres problèmes d'optimisation combinatoire. Des recherches sont menées aussi pour le développement des méthodes de résolution ainsi que l'adaptation des méthodes existantes comme les métaheuristiques. D'autre part, les recherches se sont focalisées sur la résolution des problèmes d'ordonnancement concrets rencontrés souvent dans le milieu industriel. Là où apparait l'importance des études théoriques, car elles seront exploitées pour traiter les problèmes réels.

Chaque problème d'ordonnancement est caractérisé par plusieurs contraintes en fonction du mode de traitement et de la nature de l'atelier. Parmi ces contraintes, la préemption et le transport des tâches préemptées. La préemption des tâches peut être due à plusieurs causes. Par exemple, une tâche peut être préemptée (interrompue) pour refroidir ou sécher. Un autre cas peut se présenter quand une tâche nécessite des opérations ou des traitements externes. Dans le cas de notre travail, une tâche est préemptée pour être transportée vers une autre machine où elle doit terminer son traitement.

Cette thèse porte sur l'étude d'un problème d'ordonnancement à machines parallèles identiques pour minimiser la durée totale de l'ordonnancement (*makespan*), avec des contraintes additionnelles concernant le mode de traitement et le transport des tâches. Le mode de traitement est préemptif, c'est-à-dire que n'importe quelle tâche peut être préemptée à n'importe quel moment de son exécution (traitement). De plus, la tâche préemptée va être transportée d'une machine à une autre pour poursuivre son exécution. Le changement de machine s'effectue dans un laps de temps appelé délai de transport. Ce délai de transport représente le temps nécessaire au transfert d'une tâche préemptée d'une machine à une autre, il dépend uniquement de la distance entre les machines.

L'ordonnancement sur machines parallèles avec contraintes de délais de transport a un aspect pratique, c'est l'une des raisons qui nous a poussés à l'étude de ce problème. D'autre part, plusieurs recherches tenant compte des délais de transports ont été menées à titre d'exemple les travaux de Jansen et al.[55], Lee et Chen [61], Soukhal et Martineau [83]. La plupart de ces recherches se sont concentrées sur les ateliers à machines spécialisées. Le cas des problèmes à machines parallèles en présence des délais de transport n'a pas connu le même intérêt.

La contribution de cette thèse réside dans l'apport de nouveaux résultats concernant la complexité du problème d'une part et le développement de nouvelles méthodes de résolution du problème posé d'autre part.

Le reste du document est organisé comme suit :
Dans le premier chapitre, nous rappelons les différentes notions de la complexité des algorithmes et la complexité des problèmes d'optimisation combinatoire. Puis nous donnons les définitions de bases relatives à la théorie de l'ordonnancement ainsi que les différents termes utilisés. Nous abordons aussi la classification et la réduction des problèmes d'ordonnancement entre eux. À la fin de ce chapitre, nous donnons un exposé sur les différentes méthodes de résolution et un petit rappel sur la théorie des graphes.

Dans le deuxième chapitre, nous définissons le problème puis nous présentons un état de l'art sur les problèmes d'ordonnancement à machines parallèles en tenant compte de plusieurs contraintes. Notre plus grand intérêt est porté aux problèmes d'ordonnancement avec délais de transport. Ces problèmes sont introduits à la fin de ce chapitre.

Le troisième chapitre porte sur l'étude d'un sous problème avec un nombre fixé de machines où toutes les tâches ont des durées opératoires (temps de traitement) identiques. Nous traitons le problème avec des délais de transport identiques, c'est-à-dire entre n'importe quelle paire de machines le délai de transport est le même. Puis nous traitons le problème avec délais de transport quelconques (arbitraires). Nous étudions aussi la complexité du problème lorsque le nombre de machines n'est pas fixé.

Le quatrième chapitre est dédié au problème à deux machines. En premier lieu, nous étudions la complexité du problème. Pour la résolution du problème, nous proposons un programme

dynamique et un schéma d'approximation complètement polynomial.

La résolution du problème général est abordée dans le cinquième chapitre, où une méthode en deux phases est présentée. Cette méthode consiste à résoudre le problème qui cherche la meilleure séquence des machines, dans la première phase, ce qui permettra de minimiser les délais de transport. À cet effet, nous adoptons une métaheuristique de type algorithme de colonies de fourmis. Dans la deuxième phase, l'affectation des tâches est effectuée par des heuristiques basées sur le principe des algorithmes de liste.

Une étude expérimentale est faite à la fin du quatrième et du cinquième chapitre pour évaluer la performance des algorithmes proposés. Les expérimentations se font sur des instances générées aléatoirement et l'analyse porte sur l'observation du temps moyen d'exécution ainsi que la déviation moyenne par rapport à une borne inférieure (dans le cas du problème général).

Une conclusion achève ce travail et donne une synthèse des différents résultats obtenus. Enfin quelques perspectives pour des éventuelles recherches sont proposées.

Chapitre 1

Complexité et ordonnancement

1.1 Introduction

Nous rappelons, dans ce chapitre, les principales notions utilisées tout au long du document. Ces rappels portent essentiellement sur la complexité des algorithmes, les problèmes d'optimisation combinatoire ainsi que la définition et la classification des problèmes d'ordonnancement. La fin de ce chapitre est consacrée aux différentes méthodes de résolution et un bref rappel sur la théorie des graphes.

1.2 Optimisation combinatoire

L'optimisation combinatoire est une branche de la recherche opérationnelle. Elle consiste à étudier les problèmes d'optimisation possédant un nombre fini de solutions possibles. Le but est de trouver la meilleure solution, la recherche de cette solution repose sur des méthodes bien définies.

***Définition* 1.2.1.** *[79] Un problème d'optimisation combinatoire est défini à partir d'un ensemble fini S et une application $f : S \rightarrow \Re$. Il s'agit de déterminer $\hat{s}$ tel que :*
$f(\hat{s}) = min_{s \in S} \{f(s)\}$

Le problème consistant à chercher un élément maximum au lieu d'un élément minimum est défini de la même manière puisque $max_{s \in S} \{f(s)\} = -min_{s \in S} \{-f(s)\}$.

1.3 Complexité des algorithmes

Face à un problème posé, on cherche généralement à trouver des méthodes de résolution adéquates qui permettent de donner une solution en un temps raisonnable. Là intervient la théorie de la complexité qui permet de savoir si un problème peut être résolu efficacement (en temps raisonable).
La résolution du problème posé se base sur une procédure finie et mécanique qui constitue un algorithme. Un algorithme est défini par :

***Définition* 1.3.1.** *[79] Un algorithme de résolution d'un problème (P) donné est une procédure, décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de n'importe quelle instance du problème (P) en une chaîne de caratères représentant les résultats de (P).*

Il existe une relation entre la durée d'exécution d'un algorithme et la taille de l'instance traitée. Cette relation permet de mesurer l'efficacité d'un algorithme qui s'exprime en terme du nombre d'opérations élémentaires et le nombre de caractères nécessaires pour coder les données.

***Définition* 1.3.2.** *[79] Etant données deux fonctions $f, g : \aleph \rightarrow \aleph$, on dit que f est $O(g)$ s'il existe une constante c telle que : $|f(n)| \leq c|g(n)|$ pour tout $n \in \aleph$.*
Une fonction est polynomiale si elle est $O(g)$ et si g est un polynôme en n.

***Définition* 1.3.3.** *[79] Un algorithme est polynomial si le nombre d'opérations élémentaires nécessaires pour résoudre une instance de taille n est une fonction polynomiale en n.*
Un algorithme est efficace si, et seulement si, il est polynomial.

À partir de là, on peut dire que la complexité des algorithmes consiste à étudier l'efficacité de ces derniers en se basant sur une estimation (théorique) des temps de calcul et des besoins en mémoire. Généralement on cherche à évaluer le coût des actions résultant de l'exécution d'un

algorithme, en fonction de la taille des données traitées. D'autres analyses consistent à étudier le comportement asymptotique de l'algorithme c'est-à-dire quand la taille des données n tend vers l'infini. Il y a aussi la complexité au pire des cas qui donne une borne supérieure sur le temps de calcul pour toutes les données de taille n, c'est la complexité maximum dans le cas le plus défavorable, on l'utilise quand on veut borner le temps d'exécution d'un algorithme.

1.4 Complexité des problèmes

Nous présentons dans cette section les principales notions, définitions et résultats de base s'inscrivant dans le cadre de la complexité des problèmes d'optimisation combinatoire.
Dans l'étude de la complexité des problèmes, il est important de savoir pour un problème donné s'il existe un algorithme de complexité polynomiale pour le résoudre.
Le but de la théorie de la complexité est la classification des problèmes suivant leur degré de difficulté de résolution. Pour pouvoir exposer ces différentes classes, il est nécessaire de donner quelques définitions de base.

***Définition* 1.4.1.** *[79] Un problème de reconnaissance ou de décision est un problème dont les résultats ne peuvent prendre que l'une des deux valeurs vrai ou faux.*

***Définition* 1.4.2.** *[79] Un algorithme non déterministe est un algorithme qui comporte l'instruction choix, celle-ci opérant sur un ensemble fini, choisit un élément de cet ensemble mais on ne spécifie pas à priori comment ce choix est effectué.*

Les problèmes d'optimisation combinatoire peuvent être classer suivant la complexité des algorithmes servant à les résoudre. Les classes les plus connues sont :

***Définition* 1.4.3.** *La classe P regroupe tous les problèmes pouvant être résolus par un algorithme (déterministe) polynomial. Les problèmes de la classe P sont dits faciles.*

***Définition* 1.4.4.** *La classe NP est la classe qui regroupe les problèmes de décision résolus en temps polynomial par un algorithme non déterministe.*

Parmi les problèmes de la classe NP, les problèmes NP-complets. Ces problème sont équivalents entre eux et s'il existe un algorithme polynomial pour résoudre un problème NP-complet, alors il en existe un pour tous les problème de la classe NP.
Pour décrire cette équivalence, on définit la notion de réduction polynomiale entre deux problèmes.

***Définition* 1.4.5.** *[79] Soient $P1$ et $P2$ deux problèmes de reconnaissance. On dit que $P1$ se réduit en temps polynomial à $P2$ s'il existe un algorithme pour $P1$ qui fait appel (comme à un sous-programme) à un algorithme de résolution de $P2$, et si cet algorithme de résolution de $P1$ est polynomial lorsque la résolution de $P2$ est comptabilisée comme une opération élémentaire.*

***Définition* 1.4.6.** *[79] Un problème de décision est NP-complet si tout problème de la classe NP se réduit polynomialement à lui.*

Pour chaque problème d'optimisation combinatoire, on peut lui associer un problème de décision défini comme suit :

***Définition* 1.4.7.** *[79] Etant donné un problème d'optimisation combinatoire :*
Trouver $\hat{s} \in S$ tel que $f(\hat{s}) = min_{s \in S} \{f(s)\}$ (resp. $max_{s \in S} \{f(s)\}$) et un nombre a, on définit le problème de reconnaissance ou de décision associé au problème d'optimisation :
Existe-t-il $\tilde{s} \in S$ tel que $f(\tilde{s}) \leq a$ (resp. $f(\tilde{s}) \geq a$) ?

À partir de là, on définit les problèmes NP-difficiles.

***Définition* 1.4.8.** *Un problème d'optimisation dont le problème de décision associé est NP-complet est dit NP-difficile.*

***Définition* 1.4.9.** *Un algorithme est dit pseudo-polynomial si son temps d'exécution sur une instance est borné par un polynôme en la taille des données de l'instance et par $O(p(x))$, où p un polynôme et x est une valeur dans l'instance considérée, par exemple le maximun des valeurs.*

Autrement dit, la complexité d'un algorithme pseudo-polynomial dépend des valeurs de l'instance traitée.

Remarque 1.4.1. *En ordonnancement, un algorithme est polynomial si son temps d'exécution est borné par un polynôme en fonction du nombre de tâches ou de machines. Cependant, s'il est borné par un polynôme en fonction des données de l'instance par exemple la plus grande durée opératoire correspondante à une tâche de l'instance ou bien la somme des durées opératoires de toutes les tâches, alors cet algorithme est pseudo-polynomial.*

***Définition* 1.4.10.** *Pour une instance d'un problème de décision numérique, on note $Max(n)$ la valeur du plus grand entier apparaissant dans l'instance et n la taille de l'instance.*
Pour un problème de décision π et un polynôme p, on note π_p le sous-problème de π restreint aux instances pour lesquelles $Max(n) \leq p(n)$.
Un problème NP-complet π est NP-complet au sens fort (strongly NP-complet) si seulement s'il existe un polynôme p tel que π_p est NP-complet. Sinon il est NP-complet au sens faible (NP-complet in the ordinary sense ou ordinary NP-complet).

En 1971, Cook a montré que tous les problèmes de la classe NP sont réductibles au problème de satisfiabilité d'une expression logique quelconque. C'est un problème de la logique propositionnelle (logique des prédicats). Pour le définir, rappelons quelques notions de la logique mathématique.

***Définition* 1.4.11.** *Un prédicat est défini sur un ensemble de variables logiques, à l'aide des opérations logiques élémentaires à savoir la négation (NON), la conjonction (ET) et la disjonction (OU).*
Une clause est un prédicat particulier, formée uniquement de la disjonction.
Une formule est sous forme normale conjonctive si elle s'écrit comme la conjonction de clauses.

***Définition* 1.4.12.** *Le problème de satisfiabilité noté SAT est défini par un ensemble de m clauses $\{C_1, \ldots, C_m\}$, il consiste à décider si une formule sous forme normale conjonctive $(C_1 \wedge \ldots \wedge C_m)$ est satisfiable, c'est-à-dire s'il existe une affectation des variables telle que toutes les clauses sont vraies.*

On peut énoncer maintenant le théorème de Cook :
Théorème de Cook : *Le problème de satisfiabilité est NP-complet.*

Si le problème de satisfiabilité peut être résolu en temps polynomial, alors tous les problèmes de la classe NP pourront être résolus en temps polynomial aussi.

Les preuves de NP-complétude se font généralement par réduction polynomiale à un problème déjà classé NP-complet. Garey et Johnson [44] présentent plus de 300 problèmes NP-complets.

Montrer qu'un problème de décision D est NP-complet consiste à :

1. Montrer que le problème D est dans la classe NP. Pour cela, il suffit de vérifier si une solution donnée satisfait toutes les contraintes du problème D en temps polynomial.
2. Choisir un problème D' NP-complet connu, on peut se référer à la liste établie par Garey et Johnson [44].
3. Construire une transformation f de D' à D, et ceci par la définition des instances de D' et D.
4. Montrer que la transformation f est une réduction polynomiale.

1.5 Ordonnancement

1.5.1 Définitions et notations

Dans ce qui suit, nous allons donner les différentes notions et notations liées aux problèmes d'ordonnancement, particulièrement l'ordonnancement dans les ateliers de production. Commençons par quelques définitions de base.

***Définition* 1.5.1.** *[74] L'ordonnancement est un processus de prise de décision. Il consiste à allouer des ressources à des tâches sur des périodes de temps données pour optimiser un ou plusieurs objectifs.*
Autrement dit, un problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu des contraintes temporelles et celles portant sur l'utilisation et la disponibilité des ressources requises par les tâches.

La définition d'un problème d'ordonnancement repose sur quatre notions fondamentales à savoir les tâches, les ressources, les contraintes et l'objectif à optimiser. Chacune de ces notions est définie comme suit :

- Les tâches

Une tâche est une activité dont la réalisation nécessite l'exécution d'une ou de plusieurs opérations.

Généralement, une tâche T_j est caractérisée par une une durée opératoire ou temps de traitement noté p_j (processing time), une date de disponibilité ou d'arrivée notée r_j (ready time ou release date) qui signifie que la tâche T_j ne peut pas commencer son traitement avant la date r_j et une date échue notée d_j (due date), telle que la tâche T_j doit être achevée avant cette date. Parfois un poids w_j représentant le degré d'importance de la tâche peut être associé. Ces poids sont utilisés pour créer des listes de priorité des tâches.

La résolution d'un problème d'ordonnancement permet de déterminer pour chaque tâche T_j la date de début d'exécution t_j et la date de de fin d'exécution c_j. On peut représenter une tâche par le schéma de la figure 1.1.

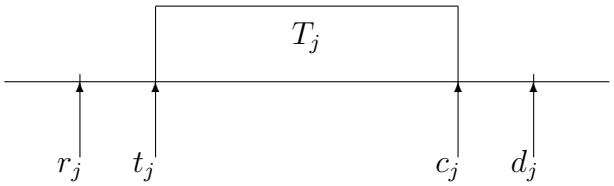

FIG. 1.1 – Caractéristiques d'une tâche traitée sur une seule machine sans préemption

- Les ressources

Une ressource est un moyen matériel ou humain mis en disposition pour la réalisation des tâche. Dans notre étude les ressources sont des machines. Une classification basée sur les différentes configurations permet de distinguer entre les modèles d'ateliers suivants :

1. Machine unique : toutes les tâches sont traitées sur une seule machine.
2. Machines parallèles : exécutent les mêmes traitements et donc peuvent traiter n'importe quelle tâche. Selon la vitesse de traitement on distingue :
   – Machines parallèles identiques qui ont toutes la même vitesse, par exemple les appareils de reprographie (photocopieurs) pocédant les mêmes caractéristiques.

- Machines parallèles uniformes, leurs vitesses sont différentes deux à deux et indépendantes des tâches.
- Machines parallèles non-liées (générales), pour lesquelles les vitesses sont différentes deux à deux et dépendent des tâches exécutées. On cite l'exemple des imprimentes à vitesses d'impression différentes. La vitesse d'impression est exprimée par le nombre de pages imprimées par minute. Le nombre de pages imprimées varie selon le type d'impression en noir et blanc ou en couleur.

3. Machines spécialisées : chaque machine est spécialisée à la réalisation de certaines opérations. Dans ce cas les tâches sont composées d'un ensemble d'opérations, chacune doit être exécutée sur une machine spécifique. Selon l'ordre de passage des opérations sur les différentes machines, on distingue entre les modes suivants :
   - Flow-shop : les opérations sont exécutées sur l'ensemble des machines dans un même ordre. L'atelier est subdivisé en étages de fabrication, chaque étage est consacré à une opération précise. Un exemple illustrant ce type d'atelier est le cas des chaînes d'assemblage des automobiles.
   - Open-shop : les opérations sont exécutées sur toutes les machines dans n'importe quel ordre.
   - Job-shop : les opérations sont exécutées sur un sous-ensemble de machines et peuvent avoir des routages (ordre de passage) différents.

D'autres modèles reflétant des problèmes industriels peuvent être définis, par exemple les ateliers de type flow-shop hybride dans lesquels un étage donné de la fabrication est assuré par plusieurs machines en parallèle.

- Les contraintes

Chaque problème d'ordonnancement est caractérisé par des contraintes qui représentent les limites imposées par l'environnement et/ou les ressources. Parmi les contraintes qui peuvent être identifiées :

- Les contraintes relatives aux dates limites des tâches, essentiellement les dates d'arrivées qui correspondent aux dates de début au plus tôt et les dates échues qui correspondent aux dates de fin au plus tard.

– Les contraintes d'antériorité décrivant les relations de précédence entre les tâches.

D'autres contraintes peuvent être définies, par exemple des contraintes décrivant le mode de traitement, la disponibilité des ressources, etc.

- L'objectif

C'est le critère d'optimisation, c'est une évaluation numérique permettant l'appréciation de la qualité des solutions. Les critères les plus courants qu'on cherche généralement à minimiser sont :

– C_{max} : *makespan*, qui vise à minimiser la date de sortie de la dernière tâche du système ou la durée totale de l'ordonnancement, $C_{max} = \max\limits_{1\leq j\leq n}\{c_j\}$.

– $\sum\limits_{j=1}^{n} w_j c_j$: somme pondérée des dates de fin d'exécution,

– L_{max} : décalage temporel maximal, tardivité ou retard algébrique. Ce critère mesure la plus grande violation des dates d'échéances souhaitées, $L_{max} = \max\limits_{1\leq j\leq n}\{L_j\} = \max\limits_{1\leq j\leq n}\{c_j - d_j\}$,

– $\sum\limits_{j=1}^{n} w_j D_j$: somme pondérée des retards, $D_j = \max\{c_j - d_j, 0\}$, qui permet d'évaluer les pénalités dues aux retards,

– $D_{max} = \max\limits_{1\leq j\leq n}\{D_j\}$: le plus grand retard,

– $\sum\limits_{j=1}^{n} w_j U_j$: somme pondérée du nombre de tâches en retard, avec

$$U_j = \begin{cases} 1 & si\ \ la\ \ tâche\ \ T_j\ \ est\ \ en\ \ retard; \\ 0 & sinon \end{cases}$$

Nous donnons également quelques définitions.

***Définition* 1.5.2.** *Un mode de traitement est dit préemptif si une tâche donnée peut être interrompue à tout instant pour terminer son exécution plus tard sans aucun coût. Dans le cas contraire, on dit que le traitement est non préemptif.*

***Définition* 1.5.3.** *Un ordonnancement est sans temps mort (ou sans arrêt) si aucune machine n'est mise en attente tant que toutes les tâches qui lui sont affectées ne sont pas encore traitées.*

Dans de nombreux problèmes d'ordonnancement classiques, des hypothèses sont posées à priori : on suppose qu'à chaque instant une machine exécute une seule tâche et une tâche est exécutée sur une machine au plus. Cependant ils existent des problèmes où ces hypothèses ne sont

pas respectées, par exemple l'ordonnancement par lot (*batch*) où une machine peut exécuter plusieurs tâches simultanément, c'est le cas des fours industriels qui traitent plusieurs pièces ayant les mêmes caractéristiques en même temps.

1.5.2 Classification

Les problèmes d'ordonnancement sont variés par leur environnement, les différentes contraintes liées aux tâches, aux ressources et au mode de traitement et par le critère à optimiser. Pour simplifier la définition et distinguer les problèmes entre eux, une classification a été proposée par Graham et al. [45]. Elle se compose de trois champs $\alpha|\beta|\gamma$ tel que :

- Le champ α est associé à l'organisation des ressources, il décrit le type et le nombre de ressources utilisées. Il comporte deux sous-champs α_1 α_2. Dans le cardre de l'ordonnancement des ateliers, α_1 représente le type des machines utilisées, il peut prendre l'une des valeurs $\{P, Q, R, F, O, J\}$. $\alpha_1 = P$ correspond aux machines parallèles identiques, $\alpha_1 = Q$ aux machines parallèles uniformes, $\alpha_1 = R$ aux machines parallèles non liées. $\alpha_1 = F$ lorsque l'atelier est composé de plusieurs machines spécialisées fonctionnant en mode flow-shop, $\alpha_1 = O$ correspond au mode open-shop et enfin $\alpha_1 = J$ pour décrire le mode job-shop.
  α_2 dénote le nombre de machines composant le système. $\alpha_2 \in \{\emptyset, m\}$, tel que $\alpha_2 = \emptyset$ si le nombre de machines est variable (non fixé) et $\alpha_2 = m$ si le nombre de machines est fixé à m.
- Le champ β décrit les différentes contraintes correspondant au mode de traitement et caractérisant les machines et les tâches. Ce champ est composé de $\{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$. β_1 décrit le mode de traitement. Lorsque $\beta_1 = pmtn$, la préemption des tâches est autorisée. β_2 caractérise les ressources supplémentaires. β_3 décrit les contraintes de précédence entre les tâches. β_4 décrit les dates de disponibilité (d'arrivée) des tâches. β_5 décrit les durées opératoires ou les temps d'exécution des tâches. Enfin, β_6 indique les dates au plus tard pour lesquelles les tâches doivent être terminées.
- Le champ γ est approprié aux critères d'optimisation, il prend l'une des valeurs : C_{max}, $\sum_{j=1}^{n} w_j c_j$, L_{max}, $\sum_{j=1}^{n} w_j D_j$, D_{max}, $\sum_{j=1}^{n} w_j U_j$ définies précédemment.

Pour mieux voir cette classification, on propose l'exemple suivant :

Exemple 1.5.1.
- $1|pmtn, r_j|\sum_{j=1}^{n} w_j c_j$: c'est le problème qui consiste à ordonnancer des tâches indépendantes sur une seule machine. Chaque tâche est caractérisée par une date d'arrivée r_j. La préemption des tâches est autorisée et l'objectif est de minimiser la somme pondérée des dates de fin de traitement.
- $Pm|prec|C_{max}$: c'est le problème avec m machines parallèles identiques, les tâches sont liées avec des contraintes de précédence. L'objectif est la minimisation de la durée totale de l'ordonnancement (makespan).
- $P2|pmtn(delay_{ii'})|C_{max}$: correspond au problème avec deux machines parallèles identiques, le mode de traitement est préemptif et il y a des délais liés au transport des tâches préemptées. L'objectif est la minimisation de la durée totale de l'ordonnancement C_{max}.

D'autre exemples illustrant les différents modes de traitement et intégrant plusieurs types de contraintes sont donnés dans [23, 74].

1.5.3 Réduction entre les problèmes d'ordonnancement

Les problèmes d'ordonnancement sont liés entre eux, l'intérêt d'étudier les différentes relations permet de faire des déductions sur l'état de complexité des problèmes d'une part. D'autre part, on peut savoir si une méthode de résolution d'un certain problème peut être appliquée pour résoudre un autre problème qui lui est réductible. Des représentations des différentes relations selon les machines utilisées, les contraintes de précédence entre les tâches, les durées opératoires des tâches ou bien selon les critères d'optimalité sont données dans [24, 45].

Nous donnons seulement le schéma représentant les relations entre les machines et les critères. Dans les deux figures 1.2 [74] et 1.3 [24, 45], le sens de la flèche indique une réduction polynomiale.

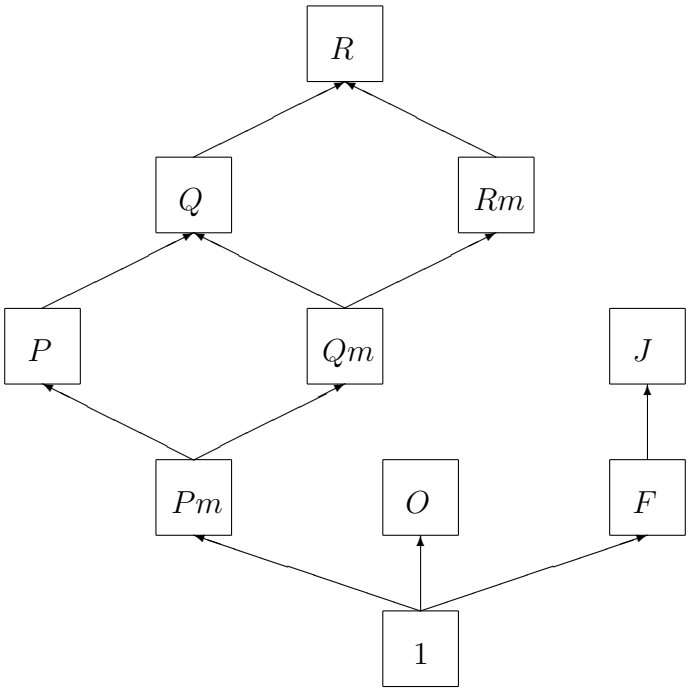

FIG. 1.2 – Réductions entre les machines

Selon la figure 1.2, les problèmes à une machine, notés (1), constituent la configuration la plus simple qui se réduit aux autres configurations avec des machines parallèles identiques (P) ou au mode de traitement open-shop (O) ou flow-shop (F). Ce dernier se réduit au mode de traitement job-shop (J).
On note aussi que les machines parallèles identiques sont un cas particulier des machines uniformes (Q) (si toutes les vitesses sont identiques). Enfin les machines uniformes sont un cas particulier des machines non-liées (R). On remarque aussi que les cas où le nombre de machines est fixé à m (Pm, Qm et Rm) se réduisent aux cas P, Q et R où le nombre de machines est non fixé. Toutes ces réductions sont faites pour un critère fixé et pour les problèmes ayant le même type de contraintes.

La figure 1.3 représente la relation entre les principaux critères. On voit bien que le critère C_{max} se réduit aux autres critères, si un problème visant la minimisation du C_{max} est NP-difficile, alors le problème de la minimisation de L_{max} l'est aussi.
Ces schémas de réduction nous permettent de tirer des conclusions sur l'état de complexité

d'un problème. Par exemple, si on dispose d'un résultat concernant un problème NP-difficile et ce problème se réduit à un autre problème général, alors on déduit que le problème général est NP-difficile. Et vice versa, si le problème général est polynomial, alors le sous problème l'est aussi.

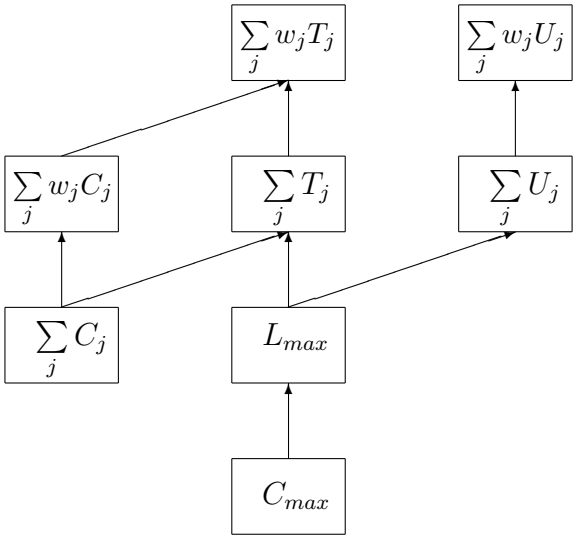

FIG. 1.3 – Réductions entre critères

1.5.4 Représentation d'un Ordonnancement

En pratique, un ordonnancement est représenté graphiquement par un diagramme de *Gantt*. Dans ce diagramme, chaque tâche est représentée par un segment horizontal de longueur proportionnelle à sa durée opératoire. On peut représenter aussi les différents temps morts, d'indisponibilité des machines, les temps de changement d'outils ou de machines, etc.

Exemple 1.5.2. *La figure 1.4, représente un diagramme de Gantt correspondant à une solution réalisable d'une instance du problème* $P3|pmtn(delay_{ii'})|C_{max}$*, cette solution a une durée égale à 13. Les durées opératoires des tâches sont données dans le tableau 1.1. La partie quadrillée représente un temps mort puisque la machine 1 reste en attente jusqu'à l'arrivée de la tâche* T_4*.*

TAB. 1.1 – Durées opératoires

T_j	T_1	T_2	T_3	T_4	T_5	T_6	T_7
p_j	2	4	4	8	7	7	4

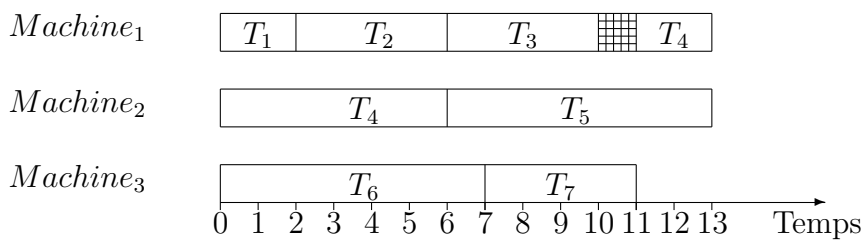

FIG. 1.4 – Diagramme de Gantt

1.6 Méthodes de résolution

Les méthodes de résolution des problèmes d'optimisation sont classées en deux grandes classes : les méthodes exactes et les méthodes approchées [43].

1.6.1 Méthodes exactes

Les méthodes de résolution exactes se caractérisent par le fait qu'elles permettent d'obtenir une ou plusieurs solutions dont l'optimalité est garantie. Parmi ces méthodes, les techniques de séparation et évaluation progressive et la programmation dynamique.

Technique de séparation et évaluation

Cette technique appelée *branch and bound* est basée sur l'énumération des solutions. Elle consiste à créer une arborescence de racine contenant l'ensemble de toutes les solutions réalisables, les autres sommets sont créés un à un lors de l'exploration. L'application de cette méthode nécessite la définition d'un principe de séparation et d'évaluation et une stratégie d'exploration.

La séparation consiste à partitionner l'ensemble des solutions contenu dans l'un des nœuds de l'arborescence en sous-ensembles. L'évaluation consiste à calculer des bornes inférieures pour les différents nœuds développés. Si l'évaluation d'un nœud est supérieure à une solution déjà obtenue dans un autre nœud, le nœud correspondant peut être stérilisé c'est-à-dire qu'on n'a pas besoin de décomposer ce nœud.
La stratégie d'exploration permet de déterminer l'ordre dans lequel on applique la séparation. On distingue deux grandes stratégies : l'exploration en profondeur d'abord et l'exploration en largeur. La première consiste à descendre dans l'arborescence jusqu'à ce qu'on trouve un sommet qu'on peut stériliser. La deuxième consiste à examiner tous les nœuds du même niveau avant de passer au niveau suivant. On peut se référer à [79] pour plus de détails sur cette méthode.

Programmation dynamique

La programmation dynamique a été introduite par Bellman dans les années 50. Elle repose sur le fait que la solution d'un problème global est obtenue en décomposant le problème en sous problèmes plus simples à résoudre. L'approche consiste donc à résoudre une série de sous problèmes jusqu'à ce qu'on trouve la solution du problème original. À chaque itération, on détermine la solution optimale pour un sous problème, en utilisant toutes les informations obtenues dans la résolution des sous problèmes précédents. Autrement dit, on commence par résoudre les plus petits sous problèmes puis on détermine les solutions des sous problèmes de plus en plus grands en se référant aux solutions déjà trouvées. Ce principe se base sur une équation récursive qui décrit la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente.
La programmation dynamique est caractérisée par :
(i) les conditions initiales ;
(ii) une relation de récurrence ;
(iii) une fonction de valeur optimale.
Des exemples d'application de cette méthode sur des problèmes d'ordonnancement classiques sont donnés par Pinedo dans [74].

1.6.2 Méthodes approchées

Etant donné un problème NP-difficile, la recherche d'une solution optimale prend beaucoup de temps car on ne connaît pas un algorithme polynomial exact pour le résoudre. Dans ce cas, les méthodes approchées sont préconisées. L'idée principale de ces méthodes consiste à chercher une solution approchée ou eventuellement exacte obtenue en temps raisonnable. Afin de gagner en temps de calcul, pour les problèmes de grande taille, ces méthodes sont appréciées même si on perd de la qualité des solutions.

Les méthodes approchées sont classées en deux classes [84] : les heuristiques et les algorithmes d'approximation ou algorithmes polynomiaux à garantie de performance. Dans ce qui suit, nous donnons quelques notions à propos des heuristiques. Cependant, la section suivante est consacrée aux algorithmes d'approximation.

***Définition* 1.6.1.** *Une heuristique est un algorithme qui a pour but de trouver une solution réalisable, tenant compte de la fonction objectif, mais sans garantie sur la qualité de la solution trouvée.*

Selon la définition donnée par Sakarovitch [79], une heuristique ou un algorithme approximatif est un algorithme qui conduit à une solution réalisable mais pas nécessairement à une solution optimale.
Les heuristiques sont à leur tour classées en deux catégories [84] : les heuristiques spécifiques et les métaheuristiques.
La première catégorie regroupe les algorithmes conçus pour résoudre un problème donné. La deuxième catégorie, les métaheuristiques sont des algorithmes qui peuvent être appliqués pour résoudre presque tous les problèmes d'optimisation (NP-difficiles). Parmi ces heuristiques : les méthodes constructives et la recherche locale.

Heuristiques constructives

Ce sont des méthodes itératives. Une solution partielle est complétée à chaque itération. Elles sont connues aussi sous l'appellation d'algorithmes gloutons. Ces algorithmes considèrent les

éléments de la solution dans un certain ordre sans jamais remettre en question un choix une fois qu'il a été fait. Pour les problèmes d'ordonnancement on peut citer les algorithmes de liste qui consistent à établir une liste selon un critère de priorité, puis construire l'ordonnancement à partir de cette liste.

Recherches locales

Ces méthodes sont initialisées par une solution réalisable obtenue par l'application d'une heuristique constructive (algorithme glouton). À chaque itération, on cherche une amélioration de la solution courante par des modifications locales jusqu'à ce qu'un critère d'arrêt soit satisfait et on ne peut plus améliorer la solution courante.

Les métaheuristiques sont apparues dans les années 80, elles s'inspirent des systèmes naturels dans de nombreux domaine : en physique (recuit simulé), en biologie de l'évolution (algorithmes évolutionnaires et génétiques) ou en éthologie (algorithmes de colonies de fourmis).
Les métaheuristiques ont connu un essor considérable depuis leur apparition. Elles sont considérées comme étant des méthodes d'approximation conçues pour la résolution des problèmes d'optimisation complexes pour lesquels les heuristiques ne donnent pas une solution efficace et les méthodes exactes prennent beaucoup de temps. On présentera les métaheuristiques les plus utilisées pour la résolution des problèmes d'ordonnancement. Des descriptions complètes de ces métaheuristiques sont données par Talbi dans [84].

Recherche tabou

La méthode de recherche tabou a été introduite en 1986 par Glover. Elle explore itérativement l'espace des solutions d'un problème en se déplaçant d'une solution courante à une nouvelle solution située dans son voisinage. Le choix de cette nouvelle solution est déterminé par l'évaluation d'une fonction objectif. Le voisinage d'une solution est l'ensemble des solutions obtenues en appliquant une transformation locale à la solution courante. Pour éviter un optimum local, une mémoire à court terme est utilisée pour conserver l'information sur le choix effectué, ceci conduit à la création d'une liste tabou. Cette information permet d'éviter certaines solutions

déjà visitées et qui pourraient la faire cycler indéfiniment dans la même région de l'espace de recherche.

Recuit simulé

Le recuit simulé a été introduit par Metropolis et al. en 1953. L'application dans le domaine de l'optimisation combinatoire revient à Kirkpatrick et al. en 1983. Cette méthode est inspirée d'un processus utilisé en métallurgie. Ce processus se base sur le principe selon lequel un système physique qui est chauffé à une très haute température et ensuite graduellement refroidi atteindra un niveau faible d'énergie correspondant à une structure moléculaire stable et forte. À chaque niveau décroissant de température, le système atteindra, après un certain temps, un état qui sera accepté automatiquement dans le cas où l'énergie du système est inférieure. Inversement, si l'énergie est supérieure, cet état sera accepté conditionnellement selon une certaine probabilité. On accepte donc une certaine dégradation de l'état du système selon certaines conditions et ces dernières deviennent de plus en plus restrictives à mesure que la température diminue. Ce processus est répété pour chaque niveau de température jusqu'à ce qu'un état solide soit atteint.

L'application de ce procédé en optimisation combinatoire se fait comme suit :
Partant d'une solution initiale qui peut être prise au hasard parmi les solutions possibles. Une énergie initiale correspond à cette solution. Cette énergie est calculée en fonction du critère à optimiser. On fait subir une modification élémentaire à la solution initiale. Cette modification entraîne une variation de l'énergie du système. Si cette variation est négative (c'est-à-dire qu'elle fait baisser l'énergie du système), elle est appliquée à la solution courante. Sinon, elle est acceptée avec une certaine probabilité. L'acceptation d'une mauvaise solution permet d'explorer une plus grande partie de l'espace des solutions et tend à éviter de s'enfermer trop vite dans la recherche d'un optimum local.

Algorithmes génétiques

Ces algorithmes ont été introduits par Holland en 1975. Ce sont des méthodes évolutives inspirées des principes de la sélection naturelle. Ils génèrent de façon aléatoire une population

initiale de solutions qui sont aussi appelés individus. Ensuite, à chaque itération de l'algorithme, on fait évoluer cette population par des mécanismes de sélection. Plusieurs techniques de sélection existent, à titre d'exemple, on cite : la sélection par rang, la sélection par roulette, la sélection uniforme, etc. Une fois que deux individus sont sélectionnés, un croisement est effectué. Ensuite, des mutations sont appliquées sur une proportion d'individus. Ce processus qui fournit une nouvelle population sera réitéré, il sera arrêté au bout d'un nombre arbitraire de générations ou lorsqu'on obtient une bonne solution.

Algorithmes de colonies de fourmis

Les algorithmes de colonies de fourmis (*Ant Colony Optimization (ACO) algorithms*) font partie d'une famille de métaheuristiques inspirées du comportement des insectes. Ils ont été introduits par Dorigo et ses collègues dans les années 90. Les algorithmes de colonies de fourmis sont inspirés de l'observation des colonies réelles. En effet, les fourmis sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid. Des expériences ont été menées par des biologistes, ils ont observé que les fourmis, ayant le choix entre deux chemins de longueurs différentes menant à une source de nourriture, avaient tendance à utiliser le chemin le plus court. Ils ont établit via leur expérience que les fourmis partagent des informations à l'aide d'une substance chimique appelée phéromone.
Le premier algorithme, appelé *Ant System* (*AS*), a été proposé par Dorigo, Colorni et Maniezzo en 1991, il consiste à construire des chemins de manière probabiliste en se basant sur la mémoire collective (trace de phéromone) pour un nombre d'itérations donné. Le mécanisme de cette métaheuristique est le suivant :
Une fourmi parcourt au hasard l'environnement autour de la colonie. Si elle découvre une source de nourriture, elle revient au nid en laissant sur son chemin une piste de phéromone. Les fourmis passant à proximité, attirées par la trace de phéromone vont suivre le même chemin en laissant à leur tour des traces de phéromone qui vont renforcer la piste de plus en plus. S'il existe deux pistes pour atteindre la même source, celle étant la plus courte est la plus parcourue, ce qui rend la trace de phéromone sur cette piste plus intense. Comme la phéromone s'évapore avec le temps, le chemin le plus long est de moins en moins emprunté. À la longue sa trace de

phéromone va disparaitre. Après un certain temps, les fourmis empruntent seulement le chemin le plus court.
Une description détaillée de l'algorithme de cette métaheuristique est donnée dans le cinquième chapitre consacré à la résolution du problème posé.

Comme il a été dit précédemment, les heuristiques n'offrent aucune garantie d'optimalité, elles peuvent trouver l'optimum pour certaines données, ou en être très éloignées pour d'autres. Supposons qu'on étudie un problème combinatoire pour lequel on dispose d'une solution optimale de référence.
Pour une heuristique H et une instance I, on note $H(I)$ la valeur de la solution heuristique et $OPT(I)$ la valeur optimale. La qualité ou la performance de la solution donnée par l'heuristique est mesurée par la distance de cette solution par rapport à la solution optimale. Ainsi, on définit la performance de l'heuristique par : $R_H(I) = \frac{H(I)}{OPT(I)}$ pour un problème de minimisation, et $R_H(I) = \frac{OPT(I)}{H(I)}$ pour un problème de maximisation.
On peut examiner la performance même si on ne dispose pas de la solution optimale pour une instance donnée I, en considérant $OPT(I)$ une borne inférieure pour un problème de minimisation et supérieure pour un problème de maximisation.

1.6.3 Algorithmes d'approximation

Dans la section précédente, nous avons évoqué la notion des méthodes approchées, en particulier les heuristiques qui donnent des solutions approchées sans aucune garantie. Maintenant, nous allons nous consacrer à la deuxième classe des méthodes approchées qui comporte les algorithmes d'approximation. Ces algorithmes, appelés aussi algorithmes polynomiaux à garantie de performance, offrent des solutions dont la qualité est estimée à priori. Pour mesurer la qualité des solutions, on peut mesurer la distance de la valeur de la solution par rapport à la valeur optimale. Cette mesure est appelée rapport d'approximation standard défini par :

***Définition* 1.6.2.** *Soit S une solution réalisable d'une instance I d'un problème P résolu par un algorithme A. Soit OPT la solution optimale sur l'instance I. Le rapport d'approximation standard de S sur I est donné par $\frac{S(I)}{OPT(I)}$.*

D'autres définitions sont aussi données :

***Définition* 1.6.3.** *[79]*
L'erreur absolue d'un algorithme A sur l'instance I est donnée par $r(A) = |S(I) - OPT(I)|$.
L'erreur relative est $R_I(A) = \frac{S(I)-OPT(I)}{OPT(I)}$ pour un problème de minimisation, et $R_I(A) = \frac{OPT(I)-S(I)}{S(I)}$ pour un problème de maximisation.

***Définition* 1.6.4.** *[79]. Soit ε un nombre positif ($0 < \varepsilon < 1$), Un algorithme A est une ε-approximation pour résoudre le problème P si $R_I(A) \leq \varepsilon$.*

Selon Demange et Paschos [32], un algorithme approché est un algorithme polynomial par rapport à la taille des instances fournissant (pour toute instance I) une solution réalisable de valeur $\lambda(I)$. La qualité de la solution est caractérisée à l'aide d'une mesure d'approximation, généralement on utilise le rapport $\gamma(I) = \min\left\{\frac{\lambda(I)}{OPT(I)}, \frac{OPT(I)}{\lambda(I)}\right\}$, où $OPT(I)$ est la valeur optimale de l'instance I. D'autres mesures peuvent être utilisées, par exemple le rapport différentiel défini par :
$\delta(I) = \frac{|\lambda(I)-\omega(I)|}{|OPT(I)-\omega(I)|}$, où $\omega(I)$ désigne la pire valeur de l'instance I. Ce rapport mesure la position, de la valeur garantie, entre la pire valeur et la meilleure valeur.

Nous introduisons dans la suite la notion d'un schéma d'approximation. Cette notion généralise la notion d'algorithme approché, son principe consiste à étudier le comportement d'une famille d'algorithmes approchés. Un schéma d'approximation peut être défini par :

***Définition* 1.6.5.** *[79] La famille d'algorithme A_ε constitue un schéma d'approximation pour la résolution du problème P si, $\forall \varepsilon > 0$, A_ε est une ε-approximation.*
Le schéma d'approximation A_ε est polynomial $PTAS$ (Polynomial Time Approximation Scheme) si pour toute valeur de ε, l'algorithme A_ε est polynomial.
Le schéma d'approximation A_ε est complètement polynomial $FPTAS$ (Fully Polynomial Time Approximation Scheme) si, pour toute valeur de ε, l'algorithme A_ε est polynomial en la taille des données et en $1/\varepsilon$.

Ce qui distingue les algorithmes d'approximation à garantie de performance des heuristiques, est la possibilité de résoudre les problèmes difficiles, mais aussi ils offrent la possibilité de

classer ces problèmes en se basant sur la propriété d'approximation et sur le type de rapport d'approximation. Les classes les plus courantes sont :

- APX : la classe des problèmes d'optimisation NP-difficiles (noté NPO) admettant un algorithme à rapport constant (ne dépend pas des paramètres de l'instance).
- $PTAS$: cette classe regroupe les problèmes NPO admettant un schéma d'approximation polynomial.
- $FPTAS$: la classe des problèmes NPO admettant un schéma d'approximation complètement polynomial.

***Définition* 1.6.6.** *Un problème de minimisation (resp. de maximisation) est dans la classe $PTAS$ si et seulement si $\forall\varepsilon$, $\exists A_\varepsilon$ un algorithme $(1+\varepsilon)$-approché (resp.$(1-\varepsilon)$-approché). S'il existe un tel algorithme sa complexité dépend de $1/\varepsilon$ et elle est polynomiale en la taille de l'instance, par contre sa complexité peut être exponentielle en $1/\varepsilon$.*

***Définition* 1.6.7.** *Un problème de minimisation (resp. de maximisation) est dans la classe $FPTAS$ s'il est dans la classe $PTAS$ et que l'algorithme est polynomial en $1/\varepsilon$. Autrement dit, un problème appartient à la classe $FPTAS$ si $\forall\varepsilon$, il existe A_ε un algorithme $(1+\varepsilon)$-approché (resp. $(1-\varepsilon)$-approché) qui soit polynomial en la taille de l'instance et en $1/\varepsilon$.*

L'idée principale pour construire un schéma d'approximation est de transformer une instance difficile du problème en une instance simplifiée. La solution optimale de l'instance simplifiée est utilisée pour calculer la solution optimale de l'instance originale. Cette approche comporte trois étapes :

- La première étape consiste à simplifier l'instance. Cette simplification porte sur la modification des données du problème.
- La deuxième étape consiste à résoudre l'instance simplifiée, la résolution peut se faire en un temps polynomial.
- Dans la troisième étape, la solution optimale de l'instance simplifiée est transformée en une solution approchée de l'instance originale et cela par le calcul de la nouvelle solution en utilisant les données de l'instance originale.

Parmi les approches utilisées pour modifier les données, nous citons :

- L'arrondissement des données en prenant la partie entière.

– La fusion par exemple dans un problème d'ordonnancement les petites tâches (de courtes durées) sont fusionnées dans les grandes tâches. Un grand nombre de petites tâches peut être pris comme une grande tâche de durée égale à la somme des durées opératoires des petites tâches.

1.7 Rappel sur la théorie des graphes

L'origine de la théorie des graphes revient aux travaux d'Euleur au $18^{ème}$ siècle. Parmi les problèmes traités à l'époque, le problème des ponts de Königsberg, le problème de la marche du cavalier sur l'échiquier et le problème du coloriage de la carte géographique. Depuis, la théorie des graphes s'est développée dans diverses disciplines telles que la chimie et la biologie.

Nous rappelons quelques définitions de base qui seront utilisées dans la suite. D'autres définitions, exemples et résultats sont détaillés dans [10, 78].

***Définition* 1.7.1.** *[78] Un graphe G est défini par :*

- *deux ensembles X et U dits ensembles de sommets et d'arcs respectivement,*
- *deux applications I et $T : U \to X$ qui associent à chaque arc son extrémité initiale et son extrémité terminale respectivement.*

D'une manière plus simple, un graphe est un schéma constitué par un ensemble fini de points et par un ensemble de liens reliant ces points.

***Définition* 1.7.2.** *[78] Un graphe est dit simple s'il ne possède pas deux arcs ayant la même extrémité initiale et la même extrémité terminale.*
Un graphe simple $G = (X, U)$ est défini par le couple constitué par :

- *un ensemble de sommets X,*
- *une partie U de $X \times X$ dite ensemble d'arcs.*

La figure 1.5 représente un graphe simple.

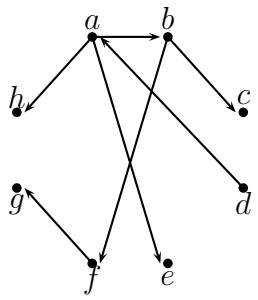

FIG. 1.5 – Graphe simple

***Définition* 1.7.3.** *[78] Un graphe non orienté $G = (X, E)$ est défini par :*

- *deux ensembles X et E dits ensembles de sommets et d'arêtes respectivement,*
- *une applications $IT : E \rightarrow X \cup P_2(X)$ qui associent à chaque arête e ses deux extrémités (qui peuvent être ideniques, auquel cas IT est à valeurs dans X et l'arête considérée est une boucle). $P_2(X)$ est l'ensemble des parties à deux élément de X.*
  Une arête dont les extrémités sont x et y est notée $e = (xy)$.

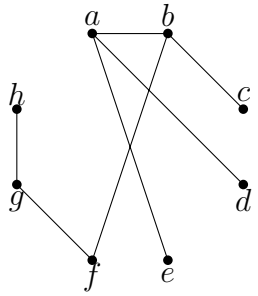

FIG. 1.6 – Graphe simple non orienté

***Définition* 1.7.4.** *Un graphe $G = (X, U)$ est dit complet si, pour toute paire de sommets (x, y), il existe au moins un arc de la forme (x, y) ou (y, x).*

***Définition* 1.7.5.** *[78] Soit $G = (X, U)$ un graphe et $x, y \in X$. Un chemin de x à y dans G est une séquence d'arcs de G telle que :*

- *l'extrémité initiale du premier arc de la séquence est x,*
- *l'extrémité initiale de chacun des autres arcs de la séquence coïcide avec l'extrémité terminale de l'arc précèdent,*
- *l'extrémité terminale du dernier arc de la séquence est y.*

***Définition* 1.7.6.** *[78]Un chemin est simple si la séquence d'arcs qui le constitu ne comporte pas plusieurs fois le même élément (arc). Un chemin est élémentaire si les sommets de G sont adjacents à deux arcs du chemin au plus.*

Autrement dit, un chemin est simple si, en le décrivant, on ne parcourt pas plusieurs fois le même arc. Un chemin est élémentaire si, en le décrivant, on ne rencontre pas plusieurs fois le même sommet.

***Définition* 1.7.7.** *[78] Un circuit est une séquence circulaire d'arcs tous distincts telle que chaque arc de la séquence soit adjacent à l'arc précèdent par son extrémité initiale et à l'arc suivant par son extrémité terminale.*
Une séquence circulaire est une séquence dans laquelle on considère que le dernier objet de la séquence est placé juste avant le premier.

***Définition* 1.7.8.** *[78] Soit $G = (X, U)$ un graphe et $x, y \in X$. Une chaîne joignant x et y dans G est une séquence d'arêtes de G telle que :*

- *la première arête de la séquence est adjacente à x par une de ses extrémités et à la seconde arête de la séquence par son autre extrémité,*
- *la dernière arête de la séquence est adjacente à y par une de ses extrémités et à l'avant dernière arête de la séquence par son autre extrémité,*
- *chaque arête intermédiaire de la séquence est adjacente à l'arête précédente par une de ses extrémités et à l'arêt suivante par l'autre extrémité.*

Les deux notions chemin simple et chemin élémentaire s'appliquent aussi sur les chaînes.

***Définition* 1.7.9.** *[78] Un cycle est une séquence circulaire d'arêtes toutes distincts telle que chaque arête de la séquence soit adjacent à l'arête précèdente par son extrémité initiale et à l'arête suivant par son extrémité terminale.*

***Définition* 1.7.10.** *[78] Un cycle (resp. une chaîne) d'un graphe $G = (X, E)$ est hamiltonien (resp. hamiltonienne) s'il (resp. elle) est élémentaire et comporte $|X|$ (resp. $(|X| - 1)$) arêtes, c'est-à-dire s'il (resp. elle) passe par tous les sommets du graphe.*

Autrement dit, un chemin hamiltonien est un chemin qui passe une fois seulement par tous les sommets du graphe G.

Il existe plusieurs applications de la théorie des graphes dans le domaine de l'ordonnancement et notament dans les problèmes avec contraintes de précédence entre les tâches ou bien les problèmes tenant compte des contraites de transport. À titre d'exemple, on cite le problème d'ordonnancement de n tâches liées par des contraintes de précédence, sous l'hypothèse que deux tâches quelconques ne peuvent pas s'exécuter simultanément sur la même machine. Ce problème se réduit au problème du chemin hamiltonien. En effet, si on construit le graphe G dont l'ensemble des sommets correspond à l'ensemble des tâches, un arc (i, j) existe si la tâche T_i précède la tâche T_j. Pour exécuter toutes les tâches en un temps minimum, il faut chercher le chemin qui passe par tous les sommets une seule fois exactement (chemin hamiltonien). La longueur du chemin donne la durée totale de l'ordonnancement.

Les problèmes de la théorie des graphes sont des problèmes de nature NP-difficile. Garey et Johnson [44] ont établient une liste contenant certain problèmes. Sakarovitch [79] a proposé quelques exemples des problèmes NP-complet relatifs à la théorie des graphes. Parmi ces problèmes,le problème du cycle hamiltonien.

1.8 Conclusion

À travers ce chapitre, nous avons abordé les éléments essentiels de la théorie de la complexité des algorithmes et la théorie de l'ordonnancement, en rappelant les principales définitions et notations qui seront utilisées dans la suite du document. Nous avons exposé aussi les différentes méthodes de résolution ainsi que quelques notions relatives à la théorie des graphes.

Chapitre 2

Position du problème et état de l'art

2.1 Introduction

Le transport joue un rôle important en industrie, que ça soit en phase de production, lorsqu'il s'agit du transport des produits semi-finis à l'intérieur de l'atelier, ou bien à la fin de la production pour transporter les produits finis vers les zones de stockage ou vers les consommateurs. Dans ce travail nous allons nous intéresser au premier type de transport dans le cas d'un atelier à machines parallèles identiques.

Dans les ateliers de production, deux modes de traitement peuvent se présenter, le traitement avec préemption et le traitement sans préemption. La préemption des tâches peut être due à plusieurs causes. Dans certains cas, une tâche a besoin de subir des traitements externes, ou bien il faut arrêter le traitement d'une tâche à un instant précis pour commencer le traitement d'une autre tâche. La tâche préemptée se poursuit sur une autre machine, ce changement de machine se fait dans un laps de temps bien déterminé, qu'on appellera délai de transport.

Le problème que nous allons étudier est un problème d'ordonnancement sur machines parallèles identiques pour la minimisation de la durée totale. Nous nous intéressons particulièrement au cas où une tâche est préemptée et transportée pour finir son traitement sur une autre machine. Nous tenons compte aussi des délais liés aux transport des tâches préemptées.

2.2 Position du problème

Considèrons le problème qui consiste à ordonnancer un ensemble de n tâches $T = \{T_1, ..., T_n\}$ indépendantes et préemptives (morcelables) de durée p_j $(j = \overline{1,\ n})$ sur m machines parallèles identiques $\{M_1, ..., M_m\}$ afin de minimiser la durée totale (C_{max}). Les contraintes du problème porte sur le mode de traitement, la préemption des tâches est autorisée. De plus, les tâches préemptées sont transportées d'une machine à une autre. Nous considérons aussi les délais de transport des tâches préemptées et transportées entre les machines. Autrement dit, si une tâche T_j traitée sur une machine M_i est préemptée pour terminer son traitement sur une autre machine $M_{i'}$, alors son transport se fait dans un laps de temps défini par le délai de transport $delay_{ii'}$. Ce délai de transport dépend uniquement de la distance entre les machines. Le transport des tâches se fait à l'aide d'un transporteur, supposé disponible et de capacité suffisante. Nous supposons que toutes les tâches sont disponibles à l'instant $t = 0$, et que toutes les machines ne présentent pas de période d'indisponibilité durant tout l'horizon de l'ordonnancement. Dans le problème défini la préemption d'une tâche sur la même machine n'est pas intéressante, il suffit de faire une permutation de la séquence de tâches traitées sur la même machine pour regrouper les parts de la tâche préemptée.
Selon la classification proposée par Graham et al. [45], ce problème est noté $P|pmtn(delay_{ii'})|C_{max}$.
Pour illustrer ce problème, nous proposons l'exemple suivant :

Exemple 2.2.1. *Considérons une instance du problème avec 3 machines et 6 tâches :* $P3|pmtn(delay_{ii'})|C_{max}$. *Les délais de transport et les durées opératoires des tâches sont donnés dans le tableau 2.1 et le tableau 2.2 respectivement.*

TAB. 2.1 – Délais de transport

	M_1	M_2	M_3
M_1	0	5	7
M_2	6	0	5
M_3	7	8	0

Le tableau 2.1 représente les délais de transport entre les différentes machines. Pour une représentation matricielle, on définit la matrice *delay* tel que l'élément $delay_{ii'}$ de cette matrice représente le temps nécessaire pour transporter une tâche de la machine M_i à la machine $M_{i'}$. On peut voir que le passage de la machine M_1 à la machine M_3 nécessite 7 unités de temps. Cette matrice n'est pas symétrique dans le cas général car le transport des tâches peut s'effectuer à l'aide d'un matériel utilisant des rails différentes (pour éviter l'encombrement dans l'atelier), ou bien des routes différentes dans le cas des ateliers multi-sites.

TAB. 2.2 – Durées opératoires

T_j	T_1	T_2	T_3	T_4	T_5	T_6
p_j	2	8	8	7	7	4

Si on considère uniquement la préemption des tâches, une solution est représentée par la figure 2.1. Cette solution n'est pas réalisable pour le problème car les contraintes de délais de transport ne sont pas respectées ($delay_{23}$=5 n'est pas respecté).

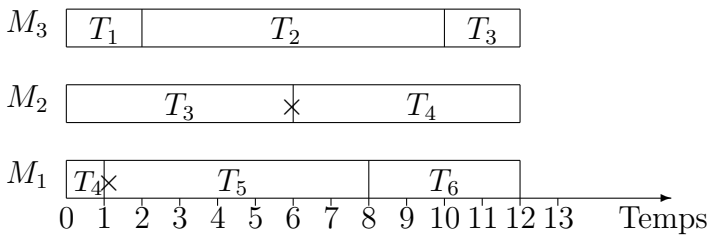

FIG. 2.1 – Représentation d'une solution sans délais de transport

Maintenant, on prend en considération les délais de transport. La solution dans ce cas est représentée par la figure 2.2. Dans cette solution, $C_{max} = 13$. On peut voir que la tâche T_3 est préemptée à l'instant $t = 6$. Elle est transportée de la machine M_2 vers la machine M_3, le délai de transport est égale à 5, il y a un temps mort sur la machine M_3 puisque la machine reste en attente jusqu'à l'arrivée de la tâche T_3.

Dans les deux figures 2.1 et 2.2, la croix indique l'instant de préemption. La flèche et la partie quadrillée de la figure 2.2 indiquent le sens du transport et le temps mort respectivement.

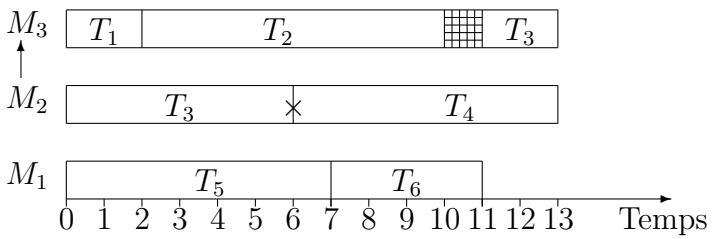

FIG. 2.2 – Représentation d'une instance du problème $P3|pmtn(delay_{ii'})|C_{max}$

2.3 Motivation

Les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire, ils trouvent des applications en pratique surtout en industrie. Le problème que nous avons définit peut se rencontrer dans un atelier de production dans le cas où des tâches non encore finies sont transportées à travers les machines. La préemption et le transport sont également rencontrés dans les problèmes d'ordonnancement des multiprocesseurs en temps réel, où la préemption d'une tâche est causée par l'arrivée d'une autre tâche au système. Dans ce cas, la tâche préemptée est transportée vers une autre machine pour poursuivre son exécution.
L'aspect de transport des tâches a fait l'objet de nombreuse recherches, la plus part se sont intéressées aux ateliers à machines spécialisées et aux problèmes d'ordonnancement en temps réel. C'est l'une des raisons qui nous a poussés à considérer et étudier le problème avec délais de transport dans le cas des ateliers à machines parallèles.

2.4 Etat de l'art

Dans ce qui suit, nous allons donner un état de l'art sur les problèmes d'ordonnancement à machines parallèles avec minimisation de la durée totale *makespan* (C_{max}). Nombreux travaux scientifiques traitant les problèmes d'ordonnancement sur machines parallèles sont publiés, nous allons citer quelque uns.

2.4.1 Problème $P|pmtn|C_{max}$

Ce problème consiste à ordonnancer un ensemble de tâches indépendantes sur des machines parallèles identiques, la préemption des tâches est autorisée. Ce problème est résolu d'une manière optimale par l'algorithme de Mc Naughton [62] en $O(n)$. L'algorithme permet de construire un ordonnancement optimal de longueur $C^*_{max} = max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}$. L'affectation des tâches consiste à charger les machines une à une jusqu'à atteindre la valeur C^*_{max} calculée au préalable. Pour cela, on sélectionne une tâche T_j de durée p_j, de l'ensemble des tâches non encore traitées, et on la traite sur la machine M_i à l'instant $t = 0$. On traite sur la même machine une autre tâche $T_{j'}$ à l'instant $(t + p_j)$, et ainsi de suite jusqu'à ce qu'on atteint C^*_{max}. Si on n'arrive pas à traiter entièrement une tâche T_j avant C^*_{max}, alors cette tâche va être préemptée, le reste de cette tâche est affecté à la machine M_{i+1}. Cet enchaînement est répété autant de fois jusqu'à ce que toutes les tâches soient traitées. Les différentes étapes de cet algorithme sont données dans l'algorithme 1.
Pour bien voir le fonctionnement de cet algorithme, nous proposons l'exemple suivant :

Exemple 2.4.1. *Considérons une instance du problème à 3 machines et 6 tâches $P3|pmtn|C_{max}$, les durées opératoires des tâches sont données dans le tableau 2.2 de la section précédante.*

La solution donnée par l'application de l'algorithme de Mc Naughton est représentée par la figure 2.3.

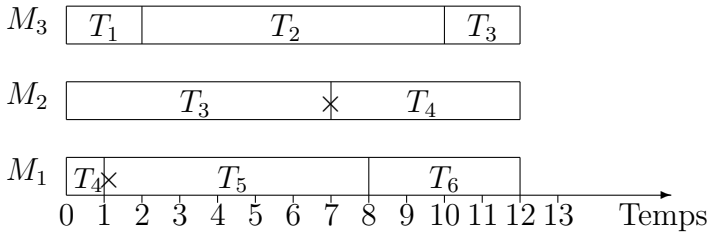

FIG. 2.3 – Solution générée par l'algorithme de Mc Naughton

La solution obtenue est de durée $C_{max} = 12$. Dans cette solution, les tâches T_3 et T_4 sont préemptées aux instants $t = 7$, $t = 1$ respectivement, ce qui est indiqué par les deux croix.

Algorithme 1 : Mc Naughton

Début

• Calculer $C^*_{max} = max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}$;

• Initialisation $t := 0$; $i := 1$; (i indice des machines)

pour chaque $j := 1$ *à* n **faire**

si $(t + p_j < C^*_{max})$ **alors**

• Affecter la tâche T_j à la machine M_i à l'instant t ;

• Poser $t := t + p_j$;

sinon

si $(t + p_j = C^*_{max})$ **alors**

• Affecter la tâche T_j à la machine M_i à l'instant t ;

• Poser $t := t + p_j$; $i := i + 1$;

sinon

• Traiter la tâche T_j sur la machine M_i à l'instant t pour $(C^*_{max} - t)$ unités de temps et le reste de la tâche $(p_j - (C^*_{max} - t))$ est affecté à la machine suivante (M_{i+1}) à $t := 0$;

fin

fin

fin

Fin.

Plusieurs contraintes peuvent s'intégrer à la contrainte de préemption. Dans [27], Cheng et Sin ont étudié le problème $P|pmtn, d_j|C_{max}$, où chaque tâche T_j a une date d'échéance d_j. En premier, ils ont passé en revue les travaux effectués sur ce type de problème, puis ils ont présenté un algorithme de résolution.

Dans [11], Berit a étudié le problème d'ordonnancement des tâches parallèles sur des machines parallèles identiques pour minimiser la durée totale. Une tâche parallèle T_j nécessite m_j machines simultanément. Berit a pris en considération les dates d'arrivées des tâches, il a étudié le problème avec et sans préemption. Des algorithmes d'approximation on été proposés.

2.4.2 Problème $P||C_{max}$

Dans ce problème, la préemption des tâches n'est pas autorisée. C'est un problème NP-difficile au sens fort vue que le nombre de machines m n'est pas fixé. Dans le cas contraire, m fixé, le problème est NP-difficile puisque le problème qui se réduit à deux machines seulement $P2||C_{max}$ l'est aussi. La preuve utilise une réduction au problème de 2-Partition [23, 74]. Pour la résolution de ce problème, une méthode exacte de type programmation dynamique a été proposée par Rothkopf [77], sa complexité est de $O(nUB^m)$ où UB est une borne supérieure pour le C_{max}. Lenstra et al. ont proposé un autre programme dynamique en $O(nUB)$ pour la résolution du problème à deux machines $P2||C_{max}$. Une autre méthode exacte a été proposée par Mokotoff [66]. Cette méthode se base sur la résolution d'un programme linéaire en variables entières et bivalentes.

Par ailleurs, plusieurs heuristiques de résolution ont été proposées. Une classe de ces heuristiques comporte les algorithmes de liste connus par *List Scheduling* (LS) et aussi sous l'appellation FAM (*First Avialable Machine*). L'algorithme consiste à affecter les tâches rangées selon un certain critère, et formant ainsi une liste de priorité, une à une dans l'ordre, à la première machine disponible. Ces algorithmes ont une performance $\frac{C_{max}(LS)}{C_{max}(OPT)} \leq (2 - \frac{1}{m})$ au plus mauvais cas [45].

Parmi les critères de priorité utilisés pour les machines parallèles, la règle LPT (*Longest Processing Time*) qui consiste à ranger les tâches dans l'ordre décroissant de leur durées opératoires. L'algorithme de liste noté algorithme FAM avec la règle LPT a une complexité de $O(nlogn)$, sa performance est $\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq (\frac{4}{3} - \frac{1}{3m})$ [45]. Les étapes de cet algorithme sont :

Algorithme 2 : FAM

Début

- Ranger les tâches dans l'ordre LPT ;
- Affecter les m premières tâches de la liste aux m machines à $t = 0$;
- Affecter les $(n - m)$ tâches restantes de la liste une à une à la première machine disponible ;

Fin

Pour illustrer le fonctionnement de l'algorithme FAM, nous l'appliquons sur les données de l'exemple 2.2.1 précédant.
Commençons par ranger les tâches dans l'ordre LPT, la liste obtenue est :
$L_{LPT} = \{T_3, T_2, T_4, T_5, T_6, T_1\}$. La solution, représentée par la figure 2.4, est de durée C_{max}=14.

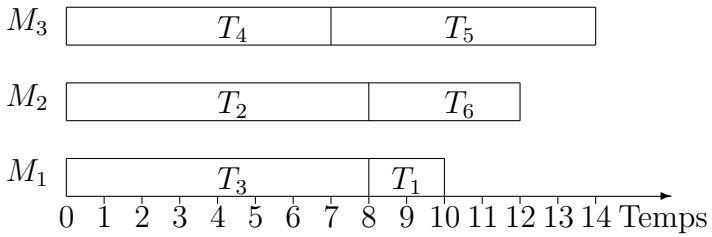

FIG. 2.4 – Solution générée par l'algorithme FAM

Un autre algorithme, *Multifit*, utilisant le principe du problème du sac à dos. L'algorithme détermine une valeur C qui représente la capacité maximale des machines. Les tâches sont affectées suivant la règle FFD (*First Fit Decreasing*) qui permet d'affecter la tâche la plus longue en respectant la capacité restante de la machine. Dans [18], Boudhar a proposé une méthode exacte de type séparation et évaluation ainsi qu'un algorithme approximatif pour la résolution du problème à machines parallèles.

D'autres chercheurs se sont intéressés à l'adaptation des métaheuristiques pour la rasolution du problème $P||C_{max}$. Nous citons particulièrement les traveaux de Min et Cheng [65] qui ont proposé un algorithme génétique.

D'autres variantes du problème ont été également étudiées en intégrant d'autres contraintes. Dans [72], Néron et al. ont étudié le problème avec des contraintes sur les dates d'arrivées des tâches. Ils ont proposé une méthode de recherche arborescente pour la résolution du problème.

Tous les problèmes précédents supposent que les machines sont disponibles durant toute la période d'ordonnancement. Ce n'est pas le cas du problème traité par Xu et al. [87]. En fait, ils ont considéré le problème pour lequel les machines ne sont pas disponibles tout le temps du fait qu'elles nécessitent une maintenance de durée variant dans l'intervalle $[T, T']$. Le problème est noté $Pm, MS[T, T']||C_{max}$, c'est un problème NP-difficile. Un autre cas similaire a été traité

par Mellouli et al. dans [63] où un problème d'ordonnancemnt sur machines parallèles avec contraintes d'indisponibilité des machines a été étudié. Dans [19], le problème avec des machines à traitement par batch a été abordé. Dans ce problème, chaque machines traite plusieurs tâches simultanément. Les tâches ayant les mêmes caractéristiques sont regroupées dans des lots par exemple des pièces qui nécessitent le même temps de cuisson et la même température, dans un four industriel, seront mise dans le même lot. Plusieurs résultats concernant la complexité du problème ont été donnés, des heuristiques de résolution et une méthode par séparation et évaluation ont été proposés.

2.4.3 Problème $P|prec|C_{max}$

Dans ce problème, les contraintes de précédence exprimant les liens existant entre les tâches sont considérées. Le problème avec des tâches de durées unitaires ($p_j = 1, \forall j$) est NP-difficile [74]. Cependant, les problèmes avec des graphes de précédence particuliers essentiellement les graphes sous forme d'arbre entrant (*in-tree*) et arbre sortant (*out-tree*) notés $P|in-tree, p_j = 1|C_{max}$ et $P|out-tree, p_j = 1|C_{max}$ respectivement sont résolus optimalement par la règle du chemin critique [74]. Le problème $P|prec, p_j = 1|C_{max}$ est également étudié dans [68] pour deux classes particulières de graphes de précédence.

D'autres contraintes telles que les contraintes des délais de communication peuvent être intégrées aux contraintes de précédence. Ces délais représentent le temps nécessaire pour transférer des données par exemple entre deux tâches liées par une contrainte de précédence et exécutées sur deux machines différentes. Ce type de problème a fait l'objet de nombreuses recherches. Dans [50], Hanen et Munier ont abordé le problème avec des petits délais de communication. Un algorithme approximatif a été proposé. Le problème avec deux processeurs identiques, tâches de durées unitaires et avec contraintes de précédence sous forme d'arbre avec de larges délais de communication a été traité par Afrati et al. [3]. Dans [53], Hoogeveen et Woeginger ont traité le problème avec tâches de durées unitaires et délais de communication unitaires aussi. Des résultats concernant la complexité du problème ont été proposés.
Engels et al. [40], se sont consacrés à l'étude du problème avec m machines parallèles identiques et des tâches de durées unitaires. Des contraintes relatives aux délais de précedence et des délais

de communication ont été considérées. Pour ce problème, le délai de précedence (*precedence delay*) représente le temps nécessaire entre l'exécution de deux tâches liées par une contrainte de précédence. Quant au délai de communication, c'est le temps nécessaire dans le cas ou les deux tâches sont traitées sur deux machines différentes.

2.4.4 Problème $P|pmtn, prec|C_{max}$

Considérons maintenant le problème avec contraintes de précédence lorsque la préemption des tâches est autorisée. Le problème est noté $P|pmtn, prec|C_{max}$ c'est un problème NP-difficile [74], deux cas particuliers $P2|pmtn, prec|C_{max}$ et $P|pmtn, tree|C_{max}$ sont résolus par l'algorithme de Muntz et Coffman en $O(n^2)$ [70]. Djellab [34] a proposé des heuristiques pour les cas particuliers suivants : $P2|pmtn, prec|C_{max}$, $P|pmtn, forest|C_{max}$ et $P|pmtn, interval - order|C_{max}$. Quelques résultats pour des classes particulières de graphes de précédence et avec des contraintes supplémentaires sont donnés par Moukrim [68].

Un autre type de relation de précédence a été défini et étudié dans [57], c'est la notion de *s*-precedence. Elle signifie que la tâche T_j ne peut commencer jusqu'à ce que la tâche T_i commence son traitement. Cette notion est différente des relations de précédences standards où la tâche T_j ne peut commencer son traitement que si la tâche T_i ait terminé.

En fin des résultats de complexité des problèmes sur machines parallèles pour plusieurs critères et en intégrant plusieurs contraintes sont donnés dans [23, 24] et [74].

Nous présentons les principaux résultats pour le critère du C_{max} lorsque la préemption des tâches est autorisée dans le tableau 2.3.

TAB. 2.3 – Problèmes avec préemption

Problème	Auteurs	Complexité
$P\|pmtn\|C_{max}$	Mc Naugthon	$O(n)$
$P\|outtree, pmtn, r_j\|C_{max}$	Lawler	$O(n^2)$
$P\|tree, pmtn\|C_{max}$	Gonzalez & Johnson	$O(nlogm)$
$Q\|pmtn, r_j\|C_{max}$	Labetoulle et al.	$O(nlogn + mn)$
$Q\|chains, pmtn\|C_{max}$	Gonzalez & Sahni	$O(n + mlogn)$
$P\|intree, pmtn, r_j\|C_{max}$	Lenstra	NP-difficile
$P\|p_j = 1, prec, pmtn\|C_{max}$	Ullman	NP-difficile
$R2\|chains, pmtn\|C_{max}$	Lenstra	NP-difficile

Le tableau 2.4, résume quelques résultats pour le critère du C_{max} lorsque la préemption des tâches n'est pas autorisée.

TAB. 2.4 – Problèmes sans préemption

Problème	Auteurs	Complexité
$P\|p_j = p, outtree, r_j\|C_{max}$	Brucker et al.	$O(n)$
$P\|p_j = p, tree\|C_{max}$	Hu	$O(n)$
$P2\|p_j = p, prec\|C_{max}$	Garey & Johnson	$O(n^{log7})$
$Q\|p_j = 1, r_j\|C_{max}$	Dessonky et al.	$O(nlogn)$
$Q2\|p_j = p, chains\|C_{max}$	Brucker et al.	$O(n)$
$P2\|\|C_{max}$	Lenstra et al.	NP-difficile
$P\|p_j = 1, intree, r_j\|C_{max}$	Brucker et al.	NP-difficile
$P\|p_j = 1, prec\|C_{max}$	Ullman	NP-difficile
$P2\|chains\|C_{max}$	Du et al.	NP-difficile
$Q\|p_j = 1, chains\|C_{max}$	Kubiak	NP-difficile

2.5 Ordonnancement avec temps de préparation

Dans cette section, nous allons nous intéresser à un autre type de contraintes, c'est les contraintes des temps de préparation.
Les temps de préparation englobent les temps nécessaires aux travaux effectués sur les machines ou sur les tâches avant de commencer le traitement proprement dit. Ces temps peuvent représenter les temps de mise en marche ou l'arrêt des machines, le démontage, l'inspection, le réglage, le changement de pièces, le nettoyage, etc. Le temps de préparation immobilise la ressource avant ou après l'exécution d'une tâche.

Une application de ce type de problème a été donnée par Bettayeb et al. [12] pour la gestion des tâches de maintenance préventive. D'autres problèmes avec ce type de contraintes ont fait l'objet de nombreuses études, en particulier les travaux de Pfund et al. [73] et Dunstall et Wirth [39]. Koulamas [58] a étudié le problème à deux machines parallèles identiques avec un serveur (robot) pour les deux machines et en considérant un temps de préparartion indépendant de la séquence des tâches. Riotteau et al. [76], ont étudié un problème avec temps de préparation dépendant de la séquence avec la contrainte *job splitting*. Contrairement à la préemption, où une part d'une tâche préemptée ne peut commencer que si l'autre part est complétée, la contrainte de *job splitting* autorise que plusieurs parts d'une même tâches puissent être exécutées simultanément. Çelik et Sariçiçek [25], Nait Tahar et al. [71], Shim et Kim [80] et Xing et Zhang [86] se sont intéressé au problème avec cette contrainte et des méthodes de résolution ont été proposées. Enfin, une classification de ces problèmes selon le type de temps de préparation a été établie par Allahverdi et al. [2], des applications ont été citées également.

2.6 Ordonnancement avec temps de transport

Dans les ateliers de production, deux types de transport peuvent se présenter. On a besoin de transporter les tâches semi-finies d'une machine à une autre pour poursuivre leur exécution ou traitement, c'est le premier type de transport. Une autre situation peut se présenter, lorsqu'on cherche à transporter les tâches finies aux entrepôts ou aux consommateurs ce qui définit le

deuxième type de transport. Plusieurs recherches traitent les modèles qui considèrent le transport des tâches, en particulier le premier type de transport qui s'effectue à l'intérieur de l'atelier de production. Nous allons citer quelques travaux effectués dans ce sens. Commençant par Lee et Chen [61] qui ont étudié un problème de flow-shop avec les deux types de transport et aussi en tenant compte des capacités des transporteurs. Soukhal et Martineau [83] ont proposé un modèle en nombres entiers et un algorithme génétique pour résoudre le problème d'ordonnancement de type flow-shop avec temps de transport. Jansen et al. [55] ont traité le cas d'un atelier job-shop lorsque le nombre m de machines et le nombre maximum d'opérations par tâches μ sont fixés. Deux versions du problème selon la contrainte de préemption ont été étudiées.
La notion du transport dans le cas des machines parallèles est traitée par Rayward-Smith [75], où les temps de transport sont définis par le terme délais de transport, ils représentent le temps nécessaire pour transférer une tâche d'une machine à une autre. Rayward-Smith a montré qu'un ordonnancement optimal peut être construit à l'aide de l'algorithme de Mc Naughton (voir algorithme 1 section 2.4.1) lorsque les délais de transport sont unitaires. Fishkin et al. [42], ont traité le problème pour des délais de transport identiques, Ils ont étudié des sous problèmes polynomiaux pour des valeurs particulières des délais.

Les notions de préemption et de transport ou migration dans le context de l'ordonnancement en temps réel sont également abordées dans de nombreuses études. Cependant, tous les résultats théoriques supposent que la préemption et la migration sont effectuées sans aucun délai. En fait, les délais sont supposés être pris en compte dans la durées opératoire de la tâche au pire cas. Dans ce contexte, une famille d'algorithmes *P-Fair* sont connus (voir [7, 30, 51]).

En fait, dans [51] deux catégories d'ordonnancement en temps réel des systèmes multiprocesseurs ont été distinguées : ordonnancemnt partitionné et ordonnancement global (*partitioning and global scheduling*). Dans le premier (*partitioning scheduling*), chaque processeur exécute les tâches de façon indépendante à partir d'une file d'attente locale. Bien que le partitionnement évite de déplacer (transporter) les tâches entre les processeurs ce qui définit la migration, l'ordonnancement global peut entraîner la migration fréquente en raison de l'utilisation d'une file d'attente partagée. Le modèle ne fait pas explicitement intégrer les délais dus à la migration des tâches. Cho et al. [28] ont étudié les modèles pour lesquels les tâches sont autorisées à migrer

arbitrairement entre les processeurs durant leur exécution (la préemption est autorisée). Ils ont supposé que le coût des changements et les délais de migration des tâches sont négligeables. Ils ont donné un nouvel algorithme d'ordonnancement optimal basé sur les algorithmes *P-Fair*. Dans [33], les auteurs ont montré que tout algorithme d'ordonnancement optimal a besoin de certaines informations sur les tâches à venir, par exemple les dates d'arrivées et dans ce cas aussi la préemption et les migrations sont effectuées sans coûts et sans prise en compte des délais.

Ainsi, comme il a été mentionné par Davis et Burns [30], les modèles étudiés ne prennent pas en considération les coûts et les délais impliqués par la préemption et la migration des tâches entre les processeurs.

Nous pouvons résumer les principales différences entre le problème d'ordonnancement que nous allons étudier dans la suite et l'ordonnancement en temps réel dans les points suivants :

– L'ordonnancement en temps réel se concentre sur les tâches qui sont périodiquement exécutées. Par conséquent, chaque tâche génère un ensemble d'opérations et chaque tâche a une date d'arrivée et une date échue.
– Il n'y a pas de fonction objectif à optimiser, il suffit de faire face au problème de faisabilité en vérifiant par exemple si chaque échéance sera respectée au moment de l'exécution.

Le problème que nous allons étudier, défini dans la section 2.2, noté $P|pmtn(delay_{ii'})|C_{max}$ a fait l'objet de quelques travaux. Nous allons citer les principaux résultats que nous avons présenté dans [22, 46] :

- Modélisation mathématique

Un modèle mathématique utilisant des variables réelles et bivalentes a été proposé.

Dans le modèle proposé, i est l'indice des tâches et j l'indice des machines. Rappelons aussi :

$c_{jj'}$: délai de transport nécessaire pour transporter une tâche de la machine M_j à la machine $M_{j'}$,

y : durée de l'ordonnancement (C_{max}) qu'on cherche à minimiser.

On définit aussi C un nombre assez grand ($C \gg 0$, on peut l'estimer à priori par $C = \sum_{i=1}^{n} p_i$).

Les variables utilisées dans le modèle sont définies par :

- La variable t_{ij} qui représente la date de début de traitement de la tâche T_i sur la machine

M_j, $t_{ij} \geq 0 \ \ \forall\ i = \overline{1,n}, \ \ \forall\ j = \overline{1,m}$.

- La variable Q_{ij} qui est la part du temps de traitement de la tâche T_i traitée sur la machine M_j, $Q_{ij} \geq 0 \ \ \forall\ i = \overline{1,n}, \ \ \forall\ j = \overline{1,m}$.

- La variable $\alpha_{iji'j'}$ définie par : $\alpha_{iji'j'} = \begin{cases} 1 & \text{si } t_{ij} \leq t_{i'j'}\,; \\ 0 & \text{sinon.} \end{cases}$

$\alpha_{iji'j'}$ exprime qu'une tâche T_i traitée sur la machine M_j commence son traitement avant la tâche $T_{i'}$ traitée sur la machine $M_{j'}$.

- La variable $x_{ij} = \begin{cases} 1 & \text{si la tâche } T_i \text{ est traitée sur la machine } M_j \\ & \text{pour } Q_{ij} \text{ unités de temps}\,;\ Q_{ij} \neq 0 \\ 0 & \text{sinon.} \end{cases}$

D'où le modèle :

$$\begin{cases} Min\ y, \\ \sum\limits_{j=1}^{m} Q_{ij} = p_i, & \forall i = \overline{1,n}, & (1)\,; \\ \mathrm{t}_{ij} + Q_{ij} \leq y, & \forall i = \overline{1,n},\ \forall j = \overline{1,m}, & (2)\,; \\ \mathrm{t}_{ij} + Q_{ij} - t_{i'j} \leq (1 - \alpha_{iji'j})C, & \forall i,\ i' = \overline{1,n} \text{ et } i \neq i', \forall j = \overline{1,m} & (3) \\ \mathrm{t}_{i'j} + Q_{i'j} - t_{ij} \leq \alpha_{iji'j}C, & \forall i,\ i' = \overline{1,n} \text{ et } i \neq i', \forall j = \overline{1,m} & (4)\,; \\ \alpha_{iji'j} + \alpha_{i'jij} = 1, & \forall i, i' = \overline{1,n} \text{ et } i \neq i',\ \forall j = \overline{1,m}, & (5)\,; \\ \mathrm{t}_{ij} + Q_{ij} + x_{ij}c_{jj'} - t_{ij'} \leq (1 - \alpha_{ijij'})C, & \forall i = \overline{1,n},\ \forall j,\ j' = \overline{1,m} \text{ et } j \neq j' & (6) \\ \mathrm{t}_{ij'} + Q_{ij'} + x_{ij}c_{j'j} - t_{ij} \leq \alpha_{ijij'}C, & \forall i = \overline{1,n},\ \forall j,\ j' = \overline{1,m} \text{ et } j \neq j' & (7)\,; \\ \alpha_{ijij'} + \alpha_{ij'ij} = 1, & \forall i = \overline{1,n},\ \forall j, j' = \overline{1,m} \text{ et } j \neq j', & (8)\,; \\ \mathrm{x}_{ij} \leq Q_{ij}C, & \forall i = \overline{1,n},\ \forall j = \overline{1,m}, & (9)\,; \\ \mathrm{x}_{ij}C \geq Q_{ij}, & \forall i = \overline{1,n},\ \forall j = \overline{1,m}, & (10)\,; \\ \mathrm{t}_{ij} \geq 0, \quad Q_{ij} \geq 0, & \forall i = \overline{1,n}, \forall j = \overline{1,m}, & \\ \alpha_{iji'j'} \in \{0,\ 1\}, & \forall i,\ i' = \overline{1,n},\ \forall j, j' = \overline{1,m} \text{ et } i \neq i',\ j \neq j', & \\ \mathrm{x}_{ij} \in \{0,\ 1\}, & \forall i = \overline{1,n},\ \forall j = \overline{1,m}, & \\ \mathrm{y} \geq 0. & & \end{cases}$$

- Les contraintes de type (1) expriment que la somme de toutes les parts de traitement de chaque tâche T_i doit être égale au temps de traitement (p_i) de la tâche.
- Si une tâche T_i débute son traitement à l'instant t_{ij} sur la machine M_j pour Q_{ij} unités de temps, elle doit obligatoirement se terminer avant la date y, c'est les contraintes de type (2).
- Pour deux tâches différentes T_i et $T_{i'}$ traitées sur une même machine M_j, soit la tâche T_i précède la tâche $T_{i'}$ ou bien l'inverse, donc soit $t_{ij} \leq t_{i'j}$, soit $t_{i'j} \leq t_{ij}$, ce qui est exprimé par les contraintes de type (3), (4) et (5). De plus, l'égalité (5) assure que l'une des deux inégalités (3) ou (4) soit satisfaite.
- Pour une tâche T_i préemptée, soit elle débute son traitement sur la machine M_j et se poursuit sur la machine $M_{j'}$ soit l'inverse. Donc soit $t_{ij} \leq t_{ij'}$, soit $t_{ij'} \leq t_{ij}$, ceci est exprimé par les contraintes de type (6), (7) et (8). De plus ces contraintes permettent de vérifier si les délais de transport sont respectés.
- Pour toute tâches T_i et toute machine M_j, si $Q_{ij} = 0$ alors la tâche T_i n'est pas traitée sur la machine M_j ce qui fait que x_{ij} vaut 0. Ceci est représenté par les contraintes de type (9).
- Pour toute tâches T_i et toute machine M_j, si $Q_{ij} \neq 0$ alors la tâche T_i est traitée sur la machine M_j pour Q_{ij} unités de temps, ce qui est exprimé par les contraintes (10).

Le modèle mathématique proposé comporte $nm(n+m+1)+1$ variables et $n+3nm(n+m-1)$ contraintes. Le nombre de contraintes peut être réduit à $n + nm(\frac{5(n+m)}{2} - 2)$ en négligeant les contraintes redondantes induites par les égalités de types (5) et (8).

Le modèle proposé permet de trouver une solution exacte au problème posé pour des instances de petites tailles. Ce modèle a été testé dans [22] pour des petites instances en utilisant un solveur *LINGO*. En effet, pour une instance de 6 tâches et 2 machines, il y a 109 variables et 222 contraintes, les résultats obtenus ont montré l'efficacité du modèle pour les instances testées. D'autres tests ont été effectués à l'aide du solveur *ILOG CPLEX 9.0* pour des instances de petite taille. Nous avons constaté que même pour les problèmes avec un nombre réduit de tâches et de machines le modèle généré comporte un grand nombre de variables et de contraintes et le temps de calcul d'une solution optimale est significativement grand. Par exemple, pour une instance de 8 tâches et 3 machines, le modèle comporte 289 variables et 620 contraintes. Le temps de calcul dépsse 75 minutes.

- Borne inférieure et supérieure

Des bornes inférieure et supérieure ont été proposées. Dans le cas général
$\overline{M} = max\left\{\max_{1\leq i\leq n}\{p_i\}, \frac{1}{m}\sum_{i=1}^{n} p_i\right\}$ est une borne inférieure pour le C_{max}.
Une autre borne inférieure est proposée aussi :

Théorème 2 de Boudhar et Haned [22] : *Si au moins une tâche est préemptée pour être traitée sur une autre machine,* $\overline{M'} = \min_{1\leq i\leq n}\{p_i\} + \min_{\substack{j,\ j'\\ j\neq j'}}\{d_{jj'}\}$ *est une borne inférieure.*

En effeft, si une tâche T_i traitée sur une machine M_j est préemptée et transportée sur une autre machine $M_{j'}$, le temps de transport nécessaire est $d_{jj'}$. Donc $C_{max} \geq \min_{1\leq i\leq n}\{p_i\} + \min_{\substack{j,\ j'\\ j\neq j'}}\{d_{jj'}\}$.

Donc $\overline{M'} = \min_{1\leq i\leq n}\{p_i\} + \min_{\substack{j,\ j'\\ j\neq j'}}\{d_{jj'}\}$ est une borne inférieure.

Comme borne supérieure, $\overline{S} = \sum_{i=1}^{n} p_i$, si toutes les tâches sont traitées sur la même machine, sinon :

Théorème 3 de Boudhar et Haned [22] : *Si au moins une tâche est préemptée pour être traitée sur une autre machine, alors* $\overline{S'} = \overline{M} + \max_{j,j'}\{d_{jj'}\}$ *est une borne supérieure.*

- Etude de cas polynomiaux

Des sous problèmes polynomiaux ont été identifiés et ceci selon les différentes valeurs que peuvent prendre les délais de transport. Parmi les sous problèmes identifiés, nous citons particulièrement le sous problème avec délais de transport $delay_{ii'} \leq e_{j,ii'}$, avec $e_{j,ii'} = \overline{M} - p_j$ est l'écart entre la date de fin de traitement de la tâche T_j préemptée sur la machine M_i et la date de reprise sur la machine $M_{i'}$. Nous avons le résultat :

Théorème 4 de Boudhar et Haned [22] : *Pour toute tâches préemptée* T_i*, si* $d_{jj'} \leq e_{ijj'}$*, alors la solution fournie par l'algorithme de Mc Naughton est optimale.*

En effeft, si $\overline{M}$ est la valeur de la solution trouvée par l'algorithme de Mc Naughton, alors toute tâche T_i préemptée sur la machine M_j et transportée vers une autre machine $M_{j'}$ a une date de début de traitement sur $M_{j'}$ qui n'est pas affectée par le délai de transport. Car après son transport, la tâche reste en attente jusqu'à ce que la machine $M_{j'}$ soit libre, c'est-à-dire jusqu'à ce que la machine termine le traitement de la tâche qui est en cours d'exécution. Comme $\overline{M}$ est une borne inférieure et toutes les contraintes du problème sont vérifiées, en particulier les contraintes des délais de transport, alors $C_{max} = \overline{M}$ donne la solution optimale.

- Heuristiques de résolution

Des heuristiques de résolution ont été proposées et testées [22]. Les résultats expérimentaux ont permis de conclure que les heuristiques basées sur le rangement des tâches et des machines conduisent à de bonnes solutions. Ce qui a été confirmé par une analyse comparative des solutions générées par les heuristiques et les solutions exactes pour des instances de petites tailles. Parmi ces heuristiques, l'heuristique $H2V2$ qui donne de bons résultats par rapport aux autres heuristiques et par rapport aux instances testées. Rappelons le principe de cette heuristique : Elle consiste à affecter les tâches rangées selon l'ordre LPT (*Longest Processing Time*) aux machines ordonnées de façon à minimiser les délais. Le principe d'affectation des tâches est le même que celui de l'algorithme de Mc Naughton de plus on fait un test pour vérifier si les délais de transport sont respectés, sinon on fait un décalage.
On résume les différentes étapes de cette heuristique dans l'algorithme suivant :

Algorithme 3 : $H2V2$

Début

- Ranger les machines ; (une procédure est utilisée pour trouver la bonne séquence des machines de telle sorte à minimiser les délais de transport).
- Ranger les tâches selon la règle LPT ;
- Affecter les tâches aux machines suivant l'algorithme de Mc Naughton et faire un décalage si nécessaire ;

Fin

Le problème $P|pmtn(delay_{ii'} = d)|C_{max}$ avec des valeurs identiques des délais de transport ($delay_{ii'} = d$) a été étudié par Fishkin el al. [42]. Les principaux résultats sont :

- Le problème $P|pmtn(delay_{ii'} = d)|C_{max}$ est résolu en $O(n)$ pour des valeurs particulières de d tel que $d \leq (LB - p_{max})$ avec $LB = \max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}$ une borne inférieure et $p_{max} = \max_{1\leq j\leq n}\{p_j\}$.

- Dans le cas où $d \geq (1+\varepsilon)(LB - p_{max})\ \forall \varepsilon > 0$, le problème est NP-difficile au sens fort. Un schéma d'approximation polynomial ($PTAS$) de complexité $O(m^{f(\varepsilon)} + n)$ a été proposé.

- Il a été montré que le problème à deux machines $P2|pmtn(delay_{ii'} = d)|C_{max}$ admet une solution optimale avec au plus un seul transport (migration selon l'appellation utilisée dans [42]). Le problème à trois machines $P3|pmtn(delay_{ii'} = d)|C_{max}$ admet une solution optimale avec au plus deux transports. Suite à ces deux résultats, une conjecture à été établie :

Conjecture *[42] : Pour le problème avec $m > 3$ machines, il existe un ordonnancement optimal pour le problème $Pm|pmtn(delay_{ii'} = d)|C_{max}$ avec au plus $(m-1)$ transports.*

Kozlov [60] a montré qu'il existe une solution optimale avec au plus trois transports pour le problème à quatre machines $P4|pmtn(delay_{ii'} = d)|C_{max}$.

Enfin, nous citons le problème avec des processeurs identiques sous contrainte de préemption et délais de transport, appelés *interprocessor communication delays*. Rayward-Smith [75] a étudié ce problème, il a montré que le problème avec délais de transport unitaires ($delay_{ii'} = 1$) est polynomial. Cependant, lorsque $delay_{ii'} \geq 2$ le problème de décision associé est NP-complet. Une réduction du problème de 3-Partition a été utilisée pour la preuve.

2.7 Conclusion

Dans ce chapitre, nous nous somme intéressé à la définition du problème. Un état de l'art sur les différents travaux effectués sur les problèmes d'ordonnancement à machines parallèles pour la minimisation du *makespan*, sous diverses contraintes, a été présenté. Des algorithmes de résolution de quelques problèmes classiques ont été présentés. Un grand intérêt a été donné à l'algorithme de Mc Naughton et l'algorithme de liste en particulier l'algorithme avec la liste LPT. Ces deux algorithmes seront appelés dans les chapitres qui suivent lors de la résolution du problème posé.

À la fin de ce chapitre, nous avons présenté les principaux résultats des recherches antérieures obtenues pour le problème posé et qui seront exploités dans la suite, pour établir de nouveaux résultats.

Chapitre 3

Etude du problème avec temps de traitement identiques

3.1 Introduction

Les problèmes d'ordonnancement avec des tâches de durées opératoire identiques sont connus sous le nom *Scheduling problems with equal-size jobs*. Ce type de problèmes peut se rencontrer au milieu industriel, notamment les problèmes d'ordonnancement avec traitement par lot dans les systèmes de production orientés. Un exemple pratique de ce type de problème est celui des systèmes d'emballage de shampoing dont le but est d'emballer dans des lots des bouteilles de shampoing. Afin de faciliter la production (rendre le temps d'installation des ressources négligeables par rapport aux temps de traitement), les lots sont divisés en tâches de durées de traitement identique par exemple, une demi-heure de temps d'emballage.
Plusieurs travaux scientifiques traitant ce genre de problème sont effectués et publiés, à titre d'exemple les travaux de Baptiste et Brucker [5], les auteurs ont présenté un état de l'art sur ces problèmes.

Dans les sections suivantes, nous allons étudier le problème d'ordonnancement avec machines parallèles identiques, des tâches de même durées opératoire. En premier, nous supposons que le nombre de machines est fixé et les délais de transport sont identiques. Puis, nous traitons

le cas où les délais de transport sont quelconques. Finalement, nous étudions la complexité du problème lorsque le nombre de machines est non fixé.

3.2 Etude du problème $Pm|pmtn(delay_{ii'} = d), p_j = p\ |C_{max}$

Dans cette section, nous supposons que le nombre de machines m est fixé, toutes les tâches ont la même durée opératoire $p_j = p,\ \forall j = \overline{1,n}$ (n est le nombre de tâches). Les délais de transport sont identiques aussi, $delay_{ii'} = d,\ \forall\ i \neq i'$. Selon la notation des problèmes d'ordonnancement proposée par Graham et al. [45], le problème considéré est noté par $Pm|pmtn(delay_{ii'} = d), p_j = p|C_{max}$.

Nous proposons l'algorithme $\mathcal{A}1$ qui considère deux cas. Le premier cas lorsque le nombre de tâches n est un multiple du nombre de machines m, l'ordonnancement est construit en affectant les tâches sans préemption aux machines. Dans ce cas, chaque machine traite $(\frac{n}{m})$ tâches. Le deuxième cas, lorsque le nombre de tâches n n'est pas un multiple de m, l'ordonnancement est obtenu en appliquant l'algorithme de Mc Naughton (voir algorithme 1 section 2.4.1) tout en respectant les délais de transport. Un décalage est éffectué si nécessaire, ce décalage permet de changer la date de reprise de la tâche préemptée afin de respecter le délai de transport. Si les délais sont très grands, ce qui induit à des décalages importants, il est plus intéressant de résoudre le problème sans préemption avec l'algorithme FAM (voir algorithme 2 section 2.4.2).

Les différentes étapes de l'algorithme $\mathcal{A}1$ sont données dans l'algorithme 4.

Algorithme 4 : $\mathcal{A}1$

Début

si *(n multiple de m)* **alors**

• Chaque machine traite $\frac{n}{m}$ tâches, $C_{max} = \frac{n}{m}p$;

sinon

si $(d \leq (\frac{n}{m} - 1)p)$ **alors**

• Appliquer l'algorithme de Mc Naughton, $C_{max} = \frac{n}{m}p$;

sinon

si $((\frac{n}{m} - 1)p < d < (\lceil \frac{n}{m} \rceil - 1)p)$ **alors**

• Appliquer l'algorithme de Mc Naughton et effectuer un décalage,

$C_{max} = p + d$;

sinon

• Appliquer l'algorithme FAM, $C_{max} = \lceil \frac{n}{m} \rceil p$;

fin

fin

fin

Fin.

Nous avons établi le résultat :

***Théorème* 3.2.1.** *[49] L'algorithme $\mathcal{A}1$ résout le problème $Pm|pmtn(delay_{ii'} = d), p_j = p|C_{max}$ en $O(n)$.*

Pour la preuve de ce théorème, nous utilisons les résultats des théorèmes 2 et 4 de Boudhar et Haned donnés dans [22] et présentés à la fin de la section 2.6.

Preuve.

- *Supposons que n est un multiple de m, alors il suffit de traiter les tâches sans préemption comme indiqué dans la figure 3.1. Dans ce cas $C_{max} = \frac{n}{m}p$.*
- *Supposons que n n'est pas un multiple de m et $d \leq (\frac{n}{m} - 1)p$. Montrons dans ce cas que l'algorithme de Mc Naughton donne une solution optimale.*

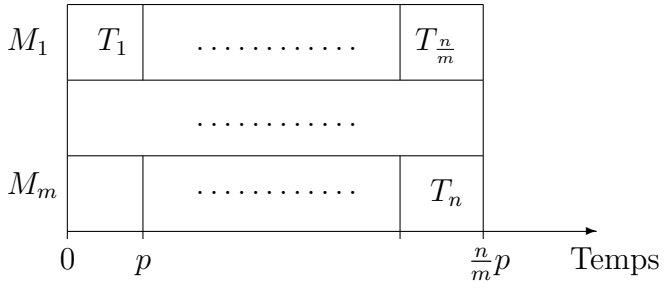

FIG. 3.1 – Cas 1 : n est multiple de m

Soit $e_{j,ii'}$ l'écart entre la date de préemption d'une tâche T_j sur la machine M_i et sa date de reprise sur la machine $M_{i'}$. Nous avons $e_{j,ii'} = \frac{n}{m}p - p = (\frac{n}{m} - 1)p, \quad \forall j, i, i'$ (voir figure 3.2). Comme $d \leq (\frac{n}{m} - 1)p$, et selon le théorème 4 de Boudhar et Haned [22] (voir section 2.6), la solution donnée par l'algorithme de Mc Naughton est optimale avec $C_{max} = \frac{n}{m}p$.

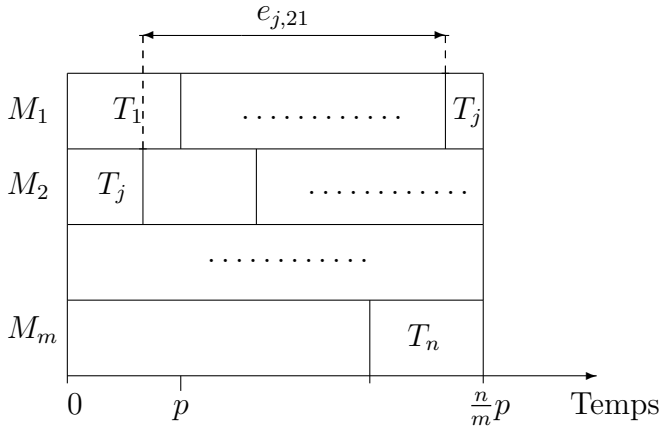

FIG. 3.2 – Cas 2 : $d \leq (\frac{n}{m} - 1)p$, n non multiple de m

– *Supposons maintenant que $(\frac{n}{m} - 1)p < d < (\lceil \frac{n}{m} \rceil - 1)p$. Si les tâches sont traitées sans préemption, alors $\lceil \frac{n}{m} \rceil p$ est une borne inférieure du C_{max}. Donc, pour obtenir une solution avec $C_{max} < \lceil \frac{n}{m} \rceil p$ certaines tâches devraient être préemptées. Par conséquent, selon le théorème 2 de Boudhar et Haned [22] (voir section 2.6) , $\min_{1 \leq j \leq n} \{p_j\} + \min_{ii', i \neq i'} \{delay_{ii'}\} = p + d$ est une borne inférieure. Ainsi, la solution obtenue en appliquant l'algorithme de Mc Naughton avec $C_{max} = p + d$ est optimal puisque $p + d < \lceil \frac{n}{m} \rceil p$ (voir figure 3.3).*
– *Supposons maintenant que $d \geq (\lceil \frac{n}{m} \rceil - 1)p$. Dans ce cas, tout ordonnancement préemptif σ conduit à une solution avec $C_{max}(\sigma) = p + d \geq \lceil \frac{n}{m} \rceil p$.*

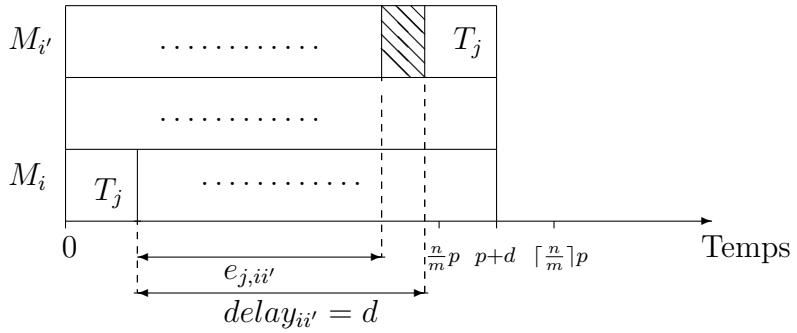

FIG. 3.3 – Cas 3 : $(\frac{n}{m} - 1)p < d < (\lceil \frac{n}{m} \rceil - 1)p$, n non multiple de m

Par conséquent, une solution optimale est donnée par le traitement des tâches sans préemption, c'est-à-dire résoudre le problème $Pm|p_j = p|C_{max}$. □

Rappelons que si le nombre de machines $m \leq 3$, il existe une solution optimale avec au plus $(m-1)$ préemptions [42]. Ainsi, nous aboutissons aux résultats suivants :

***Corollaire* 3.2.1.** *[49] Le problème à deux machines* $P2|pmtn(delay_{ii'}), p_j = p|C_{max}$ *peut être résolu de manière optimale en* $O(n)$.

Preuve. *En premier, nous déterminons l'ordre des machines* M_1 *et* M_2 *de manière à avoir le plus court délai de transport. Si* $delay_{21} \leq delay_{12}$ *(resp.* $delay_{12} \leq delay_{21}$*), alors le traitement commence sur la machine* M_1 *(resp.* M_2*) et la tâche préemptée est transportée de la machine* M_2 *(resp.* M_1*) vers la machine* M_1 *(resp.* M_2*). Nous appliquons l'algorithme* $\mathcal{A}1$*, avec l'ordre des machines trouvé dans la première étape. Dans la solution trouvée il y a au plus une tâche préemptée.* □

***Corollaire* 3.2.2.** *[49] Le problème à trois machine* $P3|pmtn(delay_{ii'}), p_j = p|C_{max}$ *peut être résolu de manière optimale en* $O(n)$.

Preuve. *Il suffit de ranger les machines* M_a*,* M_b *et* M_c *de telle sorte à avoir :*

$$\max\{delay_{ab}, delay_{bc}\} \leq \min_{l \neq k \neq j} \{\max\{delay_{jk}, delay_{kl}\}\}. \quad (3.1)$$

Dans ce cas le traitement des tâches commence sur la machine M_c. Si on atteint $\frac{n}{m}p$, le traitement des tâches se poursuit sur la machine M_b jusqu'à $\frac{n}{m}p$ et enfin les tâches restantes sont traitées sur la machine M_a. S'il y a une tâche préemptée, elle sera transportée de la machine M_a vers la machine M_b ou de la machine M_b vers la machine M_c. Dans cette solution il y a au plus deux tâches préemptées. □

Exemple 3.2.1. *Considérons une instance du problème $P3|pmtn(delay_{ii'}), p_j = p|C_{max}$ avec 3 machines. Le nombre de tâches est $n = 5$ et le temps de traitement $p = 3$. Les délais de transport sont donnés dans le tableau 3.1.*
Notons que $delay_{12} = 1$, $delay_{23} = 2$, verifient l'inégalité (3.1) précédente, d'où l'ordre des machines est : M_3, M_2, M_1. Les tâches préemptées vont être transportées de la machine M_1 à la machine M_2 ou bien de la machine M_2 vers la machine M_3.

TAB. 3.1 – Délais de transport

	M_1	M_2	M_3
M_1	0	1	3
M_2	4	0	2
M_3	5	2	0

La solution représentée par la figure 3.4 est obtenue par l'algorithme $\mathcal{A}1$.

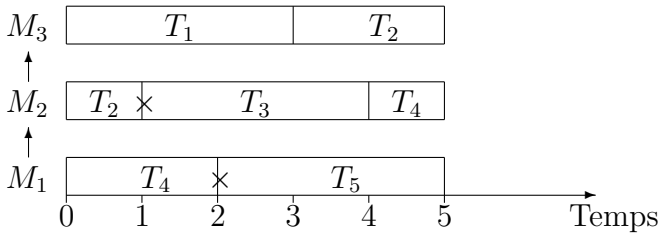

FIG. 3.4 – Solution générée par l'algorithme $A1$

Dans cette solution, la tâche T_4 est préemptée et transportée de la machine M_1 vers la machine M_2. Le délai de transport $delay_{12} = 1$ est respecté. La même remarque est faite pour la tâche T_2 qui est transportée de machine M_2 vers la machine M_3.

3.3 Etude du problème $Pm|pmtn(delay_{ii'}), p_j = p|C_{max}$

Nous supposons maintenant que seuls les temps de traitement des tâches sont identiques, les délais de transport sont quelconques. Le problème est noté $Pm|pmtn(delay_{ii'}),\ p_j = p|C_{max}$. Nous proposons un nouvel algorithme $\mathcal{A}'1$ qui consiste à déterminer la meilleure séquence des machines en premier puis affecter les tâches en suivant le même principe de l'algorithme $\mathcal{A}1$. La séquence des machines est obtenue en les rangeant dans un ordre qui permet de minimiser le maximum des délais de transport $max_{i\neq i'}\{delay_{ii'}\}$.
Les différente étapes de cet algorithme sont :

Algorithme 5 : $\mathcal{A}'1$

Début

- Chercher la séquence des machines ;
- Mettre $d = \max\limits_{i\neq i'}\{delay_{ii'}\}$;
- Appliquer l'algorithme $A1$;

Fin.

***Théorème* 3.3.1.** *L'algorithme $\mathcal{A}'1$ résout le problème $Pm|pmtn(delay_{ii'}), p_j = p|C_{max}$ en $O(f(m)+n)$.*

Preuve. *En fait, la complexité de la recherche de la meilleure séquence est en $O(f(m))$. Au pire des cas, on a $m!$ permutations pour ranger les machines. On choisit la meilleure permutation puis on applique l'algorithme $\mathcal{A}1$ en prenant $d = \max\limits_{i\neq i'}\{delay_{ii'}\}$.* □

3.4 Complexité du problème $P|pmtn(delay_{ii'}), p_j = p|C_{max}$

Dans la suite, nous allons étudier la complexité du problème lorsque le nombre de machines est non fixé. Les temps de traitement des tâches sont identiques $p_j = p,\ j = \overline{1,n}$ et les délais de transport sont quelconques. Ce problème est noté $P|pmtn(delay_{ii'}),\ p_j = p|C_{max}$.

Nous allons montrer que c'est un problème NP-difficile. La preuve est donnée par la réduction au problème de la chaîne hamiltonienne ($\mathcal{HC}$) dans un graphe $G = (V, E)$. Le problème de la

chaîne hamiltonienne est connu pour être NP-complet [44].

Une instance du problème de la chaîne hamiltonienne $\mathcal{HC}$ est définie par :

Données :

– Un graphe non-orienté $G = (V, E)$, où V est l'ensemble des m sommets et E est l'ensemble des arrêtes.

Question : Y a-t-il une chaîne qui passe par tous les sommets exactement une fois (chaîne hamiltonienne) ?

Soit $\mathcal{P}$ le problème de décision associé au problème d'ordonnancement $P|pmtn\ (delay_{ii'}), p_j = p|C_{max}$. Une instance du problème $\mathcal{P}$ est définie par :

Données :

– Un ensemble $\mathcal{N}$ de n tâches, les temps de traitement des tâches $p_j = p,\ \forall j = \overline{1,n}$, c_j est la date de fin de traitement d'une tâche T_j.

– Un ensemble $\mathcal{M}$ de m machines parallèles identiques.

– Les délais de transport sont $delay_{ii'}\ \forall (i,i') \in \{1,\ldots,m\}^2$.

– Une valeur entière Y.

Question : Y a-t-il un ordonnancement σ de $\mathcal{N}$ tel que $\max\limits_{j\in\mathcal{N}} \{c_j\} \leq Y$?

Montrons que le problème $\mathcal{HC}$ se réduit au problème $\mathcal{P}$.

À partir d'une instance quelconque du problème $\mathcal{HC}$, une instance du problème $\mathcal{P}$ est construite comme suit :

– Le nombre de machines $m = |V|$ (chaque sommet i de V est associé à une machine M_i),

– Le nombre de tâches $n = |V| + 1$,

– Les temps de traitement des tâches $p_j = p = |V|$,

– Les délais de transport sont définis par :

$$delay_{ii'} = \begin{cases} 1 & si\ \ (i,i') \in E \\ 2 & sinon. \end{cases}$$

Avec $delay_{ii'} = delay_{i'i}$ et $delay_{ii} = 0$.

– On définit aussi $Y = m + 1$.

Existe-t-il un ordonnancement de longueur inférieure ou égale à $m + 1$?

Une solution du problème $\mathcal{P}$ possède les propriétés suivantes :

***Lemme* 3.4.1.** *[49] Dans une solution du problème $\mathcal{P}$, une tâche est préemptée et transportée d'une machine à une autre au plus une fois.*

Preuve. *Considérons une solution du problème $\mathcal{P}$ avec $C_{max} \leq m+1$. Supposons qu'il existe une tâche T_k (avec $p_k = |V| = m$) préemptée et transportée plus de deux fois, dans ce cas $C_{max} \geq m+2$ puisque chaque transport nécessite un temps au moins égal à 1. Ceci est en contradiction avec $C_{max} \leq m+1$.* □

***Lemme* 3.4.2.** *[49] Dans une solution du problème $\mathcal{P}$, une tâche préemptée commence son traitement à l'instant $t = 0$ et termine à l'instant $t = m+1$.*

Preuve. *Toute tâche préemptée a une durée globale de traitement (avec délai de transport) égale à $(m+1)$, m est le temps de traitement de la tâche et 1 correspond au délai de transport. Donc, dans tout ordonnancement de durée égale à $(m+1)$, une tâche préemptée doit commencer à $t = 0$ et se termine à $t = m+1$.* □

***Lemme* 3.4.3.** *[49] Dans une solution du problème $\mathcal{P}$, le nombre de tâches traitées sur chaque machine est égal à 2.*

Preuve. *Supposons que nous ayons une solution avec $C_{max} \leq m+1$, dans laquelle il y a trois tâches T_j, T_k et T_l traitées sur la même machine M_i. Ainsi, la tâche T_j commence son traitement sur la machine M_i à $t = 0$ et la tâche T_l termine son traitement à $t = m+1$ sur la machine M_i. La tâche T_k est préemptée, elle est traitée entre les tâches T_j et T_l sur la machine M_i. Par le lemme 3.4.2, la tâche T_k commence son traitement à $t = 0$ sur une machine et se termine sur une autre machine à $t = m+1$. Cela signifie qu'elle est transportée deux fois. Donc $C_{max} \geq m+2$, contradiction.* □

***Lemme* 3.4.4.** *[49] Dans une solution du problème $\mathcal{P}$ de longueur inférieure ou égale à $(m+1)$, il y a au plus une préemption par machine.*

Preuve. *La preuve est une conséquence du lemme précédent. Considérons une solution du problème $\mathcal{P}$ avec $C_{max} \leq m+1$, dans laquelle il y a plus d'une préemption sur la machine M_i. La première tâche préemptée T_j débute à $t = 0$ sur la machine M_i et se termine sur une autre machine. La deuxième tâche préempté T_k commence après la préemption de la tâche T_j sur la machine M_i, ce qui est en contradiction avec le lemme 3.4.2. Comme il y a deux transports*

avec des délais égaux à 1, la durée totale de l'ordonnancement est supérieur ou égal à $m+2$. *Ceci est une contradiction avec* $C_{max} \leq m+1$. □

***Lemme* 3.4.5.** *[49] Dans une solution du problème* $\mathcal{P}$, *le nombre de tâches préemptées est égal à* $(m-1)$.

Preuve.

- Premièrement, supposons que le nombre de de tâches préemptées est égal à $(m+1)$. *Comme chaque tâche préemptée doit commencer son traitement à* $t=0$, *ceci contredit le résultat du lemme 3.4.2.*

- Deuxièment, supposons que le nombre de tâches préemptées est égal à m *(ces tâches débutent à* $t=0$ *sur les* m *machines). Dans ce cas on aboutira à une contradiction avec les lemmes 3.4.4 et 3.4.1 car la tâche numéro* $(m+1)$ *sera préemptée plus d'une fois.*

- Supposons maintenant que le nombre de tâches préemptées est inférieur ou égal à $(m-2)$. *Dans ce cas, il y a au moins deux tâches non préemptées qui débutent à* $t=0$ *sur deux machines différentes (soient* M_i *et* $M_{i'}$*). Donc, il reste deux morceaux de tâches de durée 1 sur chacune des deux machines (voir figure 3.5). Ce qui fait qu'il existe au moins deux tâches de durée* $(m-1)$ *qui commencent à* $t=0$ *sur deux autres machines (soient* M_k *et* $M_{k'}$*) pour lesquelles il reste deux morceaux de tâches de durée 2. On suit le même raisonnement jusqu'à ce qu'on arrive à deux tâches de durée 1 qui débutent à* $t=0$. *Si on compte le nombre de tâches, on va trouver au moins* $2+2(m-1)+2=2(m+1)>(m+1)$, *d'où la contradiction.* □

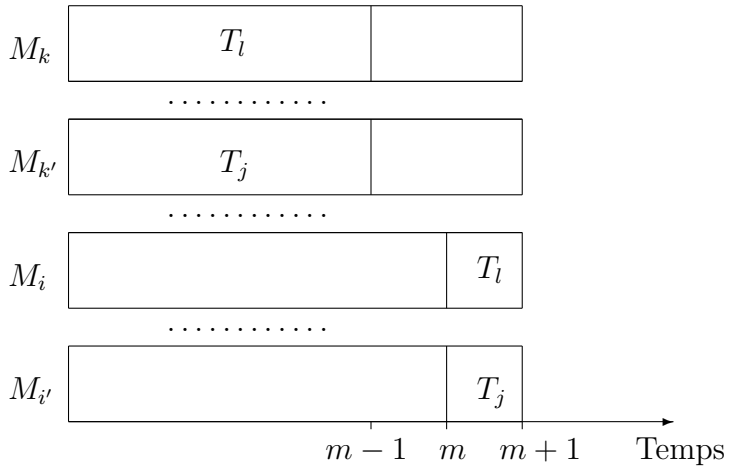

FIG. 3.5 – Cas où moins de $(m-2)$ tâches sont préemptées

***Théorème* 3.4.1.** *[49] Le problème $P|pmtn(delay_{ii'}), p_j = p|C_{max}$ est NP-difficile même si $delay_{ii'} = delay_{i'i} \in \{1, 2\}$.*

Preuve. *Considérons les deux problèmes $\mathcal{HC}$ et $\mathcal{P}$ définis précédement.*
Montrons que le problème $\mathcal{HC}$ se réduit polynomialement au problème d'ordonnancement $\mathcal{P}$.
Le problème $\mathcal{P}$ est dans la classe NP, en effet on peut vérifier en temps polynomial qu'une solution quelconque vérifie toutes les contraintes.

Supposons que le problème $\mathcal{HC}$ admet une solution. Donc, il existe une chaîne hamiltonienne dans le graphe $G = (V, E)$. À partir de cette solution nous pouvons construire l'ordonnancement cherché en prenant les machines dans l'ordre inverse des sommets constituant la chaîne (M_m, $M_{m-1}, \ldots, M_1$). Les tâches sont traitées sur les différentes machines suivant le principe de l'algorithme de Mc Naughton. S'il y a une tâche préemptée T_j, elle sera placée à la première position sur la machine M_i et à la dernière position sur la machine M_{i+1}. Autrement dit, nous commençons par remplir la machine M_{i+1} puis la machine M_i et ainsi de suite. S'il y a une tâche préemptée, elle sera traitée à la première position sur la machine M_i et à la dernière position sur la machine M_{i+1}. Le transport se fait de la machine M_i vers la machine M_{i+1}. De cette façon l'écart entre la date de préemption et la date de reprise est égal à 1 et le délai de transport $delay_{ii+1}$(qui est égal à 1 dans ce cas) est respecté. Il s'ensuit que $C_{max} \leq |V| + 1$ (voir figure 3.6).

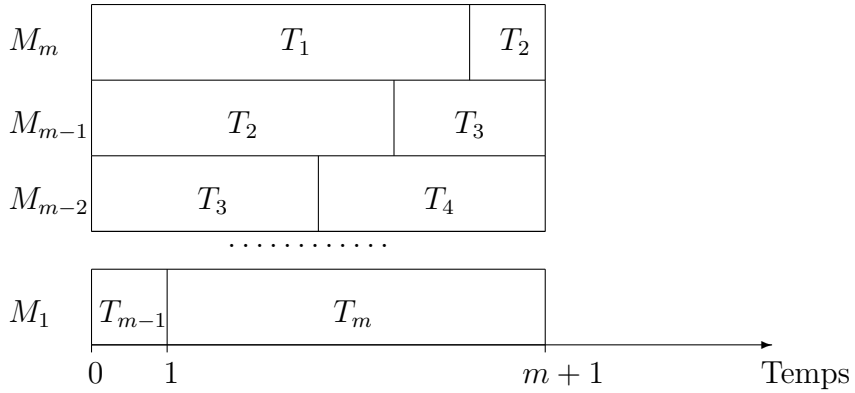

FIG. 3.6 – Solution du problème $\mathcal{P}$

Supposons maintenant que nous ayons une solution du problème d'ordonnancement avec une durée inférieure ou égale à $|V|+1$. *Cette solution possède les propriétés montrées dans les lemmes 3.4.1, 3.4.2, 3.4.3, 3.4.4 et 3.4.5 précédents.*

La solution est de durée égale à $(m+1)$. *Il existe une tâche qui débute à* $t=0$ *est se termine à* $t=m$ *sur une machine* M_i *(sans perte de généralité, nous pouvons supposer que* $i=m$*). Le morceau de tâche restant est de durée égale à 1. Donc, il existe une tâche de durée* $(m-1)$ *qui commence à* $t=0$ *sur une machine* $M_{i'}$ *(sans perte de généralité, nous pouvons supposer que* $i'=m-1$*). Le morceau de tâche qui reste est de durée égale à 2. D'où, il existe une tâche de durée* $(m-2)$ *qui commence à* $t=0$ *sur la machine* M_{i-2}*, il y a une tâche de durée* $(m-3)$ *qui commence à partir de* $t=0$ *sur la machine* M_{i-3}*. Et ainsi de suite jusqu'à la machine* M_1 *sur laquelle une tâche de durée 1 débute à* $t=0$ *et une autre tâche de durée* m *commence à* $t=1$ *et termine à* $t=m+1$.

L'écart entre la date de préemption et la date de reprise de chaque tâche est égal à 1, donc les délais de transport sont respectés.

Une chaîne hamiltonienne est obtenue en prenant les sommets dans l'ordre des machines (c'est-à-dire $M_1, M_2, \ldots, M_m$*). Donc, le problème* $\mathcal{HC}$ *admet une solution.* □

Nous aboutissons au résultat : le problème d'ordonnancement $P|pmtn(delay_{ii'}), p_j = p|C_{max}$ est NP-difficile.

3.5 Conclusion

À travers ce chapitre, nous avons étudié un cas particulier du problème $P|pmtn(delay_{ii'})\ |C_{max}$, lorsque toutes les tâches ont la même durée opératoire. Nous avons montré que le problème avec un nombre de machines fixé, lorsque les délais de transport sont identiques ou quelconques, est polynomial. Cependant, le problème avec un nombre non fixé de machines est NP-difficile. En effet, même si les tâches ont des durées opératoire identiques le problème est NP-difficile car il se réduit au problème de la chaîne hamiltonienne qui est lui même un problème NP-difficile.

Chapitre 4

Etude du problème à deux machines $P2|pmtn(delay_{ii'})|C_{max}$

4.1 Introduction

Dans ce chapitre, nous allons nous intéresser particulièrement au problème à deux machines. Fishkin et al. [42] ont montré qu'il existe un ordonnancement optimal avec au plus un transport (migration selon leur appellation) pour le problème à deux machines avec délais de transport identiques $P2|pmtn(delay_{ii'} = d)|C_{max}$. Ils ont montré aussi que le problème à trois machines et délais de transport identiques $P3|pmtn(delay_{ii'} = d)|C_{max}$ admet un ordonnancement optimal avec au plus deux transports (migrations).
Kozlov [60], a étudié le problème à quatre machines $P4|pmtn(delay_{ii'} = d)|C_{max}$, il a montré qu' il existe un ordonnancement optimal avec au plus trois transports (migrations).

Dans ce qui suit, nous considèrons le problème à deux machines avec délais de transport quelconques $P2|pmtn(delay_{ii'})|C_{max}$, c'est un problème NP-difficile [22]. Une nouvelle preuve sera présentée dans la section suivante. Par la suite, la résolution du problème se fait à l'aide d'un programme dynamique. Nous allons montrer aussi que ce problème admet un schéma d'approximation complètement polynomial ($FPTAS$).

4.2 Complexité du problème à deux machines

Considérons le problème $P2|pmtn(delay_{ii'} = d)|C_{max}$, nous avons le résultat suivant :

***Théorème* 4.2.1.** *[22] Le problème $P2|pmtn(delay_{ii'} = d)|C_{max}$ est NP-difficile.*

Nous avons présenté une preuve dans [22] qui utilise une réduction au problème NP-complet de 2-Partition [44].

Nous proposons une autre preuve pour le théorème 4.2.1. Cette nouvelle preuve se base aussi sur la réduction au problème de partition qui est NP-complet [44].

Preuve. *Soit $\mathcal{P}$ le problème de décision associé au problème d'ordonnancement $P2|pmtn(delay_{ii'} = d)|C_{max}$. Le problème $\mathcal{P}$ est défini par :*

***Données** :*

- *Un ensemble $\mathcal{N}$ de n tâches de durées p_j, c_j correspond à la date de fin de traitement d'une tâche T_j, $\forall j = \overline{1,n}$.*
- *Les délais de transport $delay_{12} = delay_{21}$,*
- *Un entier Y.*

***Question** : Existe-il un ordonnancemnt σ de l'ensemble $\mathcal{N}$ tel que $\max_{j\in\mathcal{N}} \{c_j\} \leq Y$?*

Le problème de partition est défini par :

***Données** :*

- *Un ensemble fini $\mathcal{I}$ de n éléments $a_1, a_2, \ldots, a_n$, de tailles entières $s(a_i)$, $\forall i = \overline{1,n}$, avec $\sum_{i=1}^{n} s(a_i) = 2B$.*

***Question** : Existe-il un sous-ensemble $\mathcal{I}_1$ des indices tel que $\sum_{i\in\mathcal{I}_1} s(a_i) = \sum_{i\in\mathcal{I}\setminus\mathcal{I}_1} s(a_i) = B$?*

Nous montrons que le problème de partition se réduit au problème $\mathcal{P}$.

Etant donnée une instance quelconque du problème de partition, une instance du problème d'ordonnancement $\mathcal{P}$ est construite comme suit :

- $\mathcal{N} = \{1, 2, ..., n+1\}$,
- *pour $j \in \{1, 2, ..., n\}$: $p_j = s(a_j)$; $p_{n+1} = 2$. Sans perte de généralité, nous supposons que $s(a_j) \geq 3$ $\forall j$,*

– $delay_{12} = delay_{21} = B - 1$,

– $Y = B + 1$,

La solution du problème $\mathcal{P}$ est obtenue par l'ordonnancement du sous-ensemble de tâches correspondantes à l'ensemble $\mathcal{I}_1$ sur la machine M_1. La tâche $(n+1)$ sera préemptée après une unité de temps de son exécution. Les tâches correspondantes à l'ensemble $\mathcal{I} \backslash \mathcal{I}_1$ seront exécutées avant la tâche préemptée $(n+1)$, la deuxième part de la tâche $(n+1)$ commence à l'instant B et se termine à l'instant $(B+1)$. Cet ordonnancement satisfait les conditions du problème et par la suite, le problème $\mathcal{P}$ a une solution.

Supposons maintenant qu'il existe une solution du problème d'ordonnancement $\mathcal{P}$, alors il existe un ordonnancemnt tel que $\max_{j \in \mathcal{N}} \{c_j\} \leq Y$. Donc :
- La tâche $(n+1)$ est partagée en deux parts égales, la première part est exécutée sur la machine M_1 à la première position pour une unité de temps, la deuxième part de la tâche est exécutée sur la machine M_2 dans l'intervalle de temps $[B, B+1]$. Il est claire qu'il n'y a aucune tâche qui peut finir son traitement avant $B+2$, sinon cette tâche sera préemptée et transportée.
- Il y a k tâches ordonnancées sans préemption sur la machine M_1 dans l'intervalle de temps $[1, B+1]$. Les indices de ces tâches vont définir le sous-ensemble $\mathcal{N}_1$. Donc :

$$\sum_{j \in \mathcal{N}_1} p_j + 1 \leq Y \Leftrightarrow \sum_{j \in \mathcal{N}_1} s(a_j) \leq B$$

$$\Leftrightarrow \sum_{j \in \mathcal{N} \backslash (\mathcal{N}_1 \cup \{n+1\})} p_j \geq B.$$

Comme $\sum_{j \in \mathcal{N} \backslash \{n+1\}} s(a_j) = 2B$, *alors* $\sum_{j \in \mathcal{N}_1} s(a_j) = B$ *et* $\sum_{j \in \mathcal{N} \backslash (\mathcal{N}_1 \cup \{n+1\})} s(a_j) = B$.
Par la suite le problème de partition admet une solution. □

Le problème $P2|pmtn(delay_{ii'} = d)|C_{max}$ est NP-difficile. Par conséquent, le problème $Pm|pmtn$ $(delayii')|C_{max}$ est NP-difficile.

4.3 Programme dynamique pour le problème $P2|pmtn(delay_{ii'})|C_{max}$

Nous proposons maintenant une méthode de résolution au problème à deux machines $P2|pmtn(delay_{ii'})|C_{max}$.

L'approche utilisée s'appuie sur un programme dynamique permettant de résoudre le problème $P2||C_{max}$, sans préemption et sans tenir compte des délais de transport avec $(n-1)$ tâches. À chaque fois une tâche T_j $(j = \overline{1,n})$ est négligée. Nous allons exploiter aussi le résultat donné dans [42] qui caractérise un ordonnancement optimal. À partir de là, l'ordonnancement optimal du problème $P2|pmtn(delay_{ii'})|C_{max}$ est obtenu en ajoutant la tâche négligée. Cette tâche sera préemptée et traitée sur les deux machines, de sorte qu'elle soit à la première position sur une machine et à la dernière position sur l'autre machine. Ce procédé est appliqué successivement n fois afin de déterminer la meilleure tâche qui sera préemptée, ce qui correspond à la solution optimale.

Le programme dynamique que nous allons proposer est basé sur la définition de la fonction $F(n, P_1)$ qui détermine le temps minimum de l'ordonnancement d'un ensemble de n tâches tel que la somme des temps de traitement sur la machine M_1 est égale à P_1. Il est claire que $UB = \sum_{j=1}^{n} p_j$ est une borne supérieure pour P_1.

La décision consiste à affecter une tâche à la machine M_1 ou M_2.

Considérons le programme dynamique DP pour résoudre le problème classique $P2||C_{max}$. Ce programme dynamique se base sur la fonction F suivante :

$$F(j,P_1) = \begin{cases} 0 & j = 0 \ \ et \ \ P_1 \geq 0 \\ \sum_{k=1}^{j} p_k & j > 0 \ \ et \ \ P_1 = 0 \\ +\infty & \forall\ j > n \ et \ \forall\ P_1 \\ +\infty & \forall\ j \in \{0,1,\ldots,n\} \ \ et \\ & P_1 < 0 \ ou \ P_1 > UB \\ min\{F(j-1,P_1-p_j), F(j-1,P_1)+p_j\} & \forall\ j \in \{0,1,\ldots,n\} \ \ et \\ & \forall\ 0 \leq P_1 \leq UB \end{cases}$$

La fonction F permet de calculer la valeur de la fonction objectif C_{max} et de déterminer l'affectation des tâches. Dans la formule récursive, $F(j, P_1) = F(j-1, P_1 - p_j)$ correspond à l'affectation de la tâche T_j à la machine M_1, et $F(j, P_1) = F(j-1, P_1) + p_j$ correspond à l'affectation de la tâche T_j à la machine M_2.

La valeur optimale du C_{max} est déterminée par :

$$\mathrm{C}_{max}(Opt) = \min_{\frac{UB}{2} \leq P_1 \leq UB} \{max\{F(n, P_1), P_1\}\}. \quad (4.1)$$

La solution est obtenue en utilisant une procédure de retour arrière (*backtracking*).

La complexité de cet algorithme est en $O(nUB)$ qui est meilleure que la complexité de l'algorithme du programme dynamique proposé par Rothkopf [77] où une solution optimale est obtenue en $O(nUB^2)$.
Pour mieux voir le fonctionnement du programme dynamique pour la résolution du problème $P2||C_{max}$, nous proposons l'exemple suivant :

Exemple 4.3.1. *Considérons une instance du problème $P2||C_{max}$ avec 6 tâches ($n = 6$), les durées opératoires des tâches sont donnés dans le tableau 4.1.*

TAB. 4.1 – Durées opératoires

T_j	T_1	T_2	T_3	T_4	T_5	T_6
p_j	2	1	3	1	1	4

La fonction F est représentée dans le tableau 4.2, les colonnes représentent les différentes valeurs de P_1 et les lignes donnent les valeurs de j.
La valeur du C_{max}, déterminée à l'aide de l'égalité (4.1) précédente, est $C_{max} = 6$.
Pour avoir la séquence des tâches, nous utilisons une procédure de retour arrière (les valeurs de la fonction F utilisées à ce fait sont en gras).
Les tâches T_6, T_5 et T_4 sont traitées sur la machine M_1. Les tâches T_3, T_2 et T_1 sont traitées sur la machine M_2.

$j \setminus P_1$	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	**2**	2	0	0	0	0	0	0	0	0	0	0	0
2	**3**	2	1	0	0	0	0	0	0	0	0	0	0
3	**6**	5	4	3	2	1	0	0	0	0	0	0	0
4	7	6	5	4	3	2	1	0	0	0	0	0	0
5	8	7	**6**	5	4	3	2	1	0	0	0	0	0
6	12	11	10	9	8	7	**6**	5	4	3	2	1	0

TAB. 4.2 – La fonction $F(j, P_1)$

Maintenant nous considérons l'algorithme DPX (voir algorithme 6). Nous avons le résultat :

***Théorème* 4.3.1.** *[49] Le problème $P2|pmtn(delay_{ii'})|C_{max}$ est résolu en $O(n^2UB)$ par l'algorithme DPX.*

Preuve. *Pour déterminer la solution optimale, il suffit de choisir une tâche T_j et de l'enlever de l'ensemble des tâches. Nous appliquons le programme dynamique DP sur les $(n-1)$ tâches restantes. Nous rajoutons la tâche T_j à la solution obtenue en la plaçant à la première (resp. dernière) position sur la machine M_2 (resp. M_1). Nous verifions si le délai de transport est respecté. Sinon un décalage est effectué pour changer la date de reprise de la tâche sur l'une des deux machines. De cette façon, la tâche T_j est placée sur les deux machines de telle sorte à minimiser C_{max}.*

Cette opération est répétée n fois pour avoir le meilleur ordonnancement avec exactement une préemption. Cet ordonnancement est comparé avec l'ordonnancement obtenu en appliquant le programme dynamique DP avec les n tâches sans préemption (pour résoudre le problème $P2||C_{max}$) afin de choisir le meilleur.

Ainsi, une solution optimale est calculée en $O(n^2UB)$. □

Algorithme 6 : DPX

Début

• Calculer $LB = max\left\{\frac{1}{2}\sum_{j=1}^{n} p_j, \max_{1\leq j\leq n}\{p_j\}\right\}$.

• Résoudre le problème $P2\,||\,C_{max}$ par l'algorithme FAM (algorithme 2 section 2.4.2), la solution obtenue est C_{max}^{LPT}.

Soit T_k la dernière tâche ordonnancée, c'est-à-dire sa date de fin de traitement est égale C_{max}.

si *($C_{max}^{LPT} = LB$)* **alors**

| L'algorithme FAM donne la solution optimale.

fin

si *($(LB - p_k) \geq \max_{i,i'}\{delay_{ii'}\}$)* **alors**

| La solution obtenue par l'algorithme FAM en préemptant la tâche T_k pour avoir $C_{max} = LB$, est optimale.

fin

si *($\min_{i,i'}\{delay_{ii'}\} \geq LB$)* **alors**

| Le programme dynamique DP appliqué au problème $P2||C_{max}$ donne la solution optimale (DP est appliqué en prenant $P = C_{max}^{LPT}$).

fin

• Résoudre le problème $P2||C_{max}$ en utilisant le programme dynamique DP avec $P = C_{max}^{LPT}$.

pour $j := n$ ***bas*** 1 **faire**

| • Utiliser le programme dynamique DP pour trouver la solution optimale S_j de valeur f_j pour le problème $P2||C_{max}$ sans la tâche T_j
| • Chercher la meilleure solution, pour le problème $P2|pmtn(delay_{ii'})|C_{max}$, parmi les solution trouvées avec $(n-1)$ tâches pour le problème $P2||C_{max}$. La tâche T_j est partagée en deux parts qui seront placées sur le deux machines de façon à minimiser le délai de transport.

fin

• Choisir la meilleure solution parmi les n solutions trouvées dans la boucle et la solution du problème $P2||C_{max}$ avec n tâches.

Fin.

Remarque 4.3.1. *Comme dans le cas de trois machines, il existe un ordonnancement optimal avec au plus deux transports [42]. Une solution optimale pour le problème* $P3|pmtn\ (delay_{ii'} = d)|C_{max}$ *peut être trouvé en utilisant la même idée de l'algorithme* DPX*. Dans ce cas, il faut déterminer les deux tâches qui vont être préemptées et transportées. Pour résoudre le problème sans préemption avec* $(n-2)$ *tâches, le programme dynamique de Rothkopf [77] peut être appliqué. Cet algorithme donne une solution optimale en* $O(nUB^3)$*.*
Ce principe peut s'étendre au problème à quatre machines $P4|pmtn\ (delay_{ii'} = d)|C_{max}$*. En effet, suite au résultat présenté par Kozlov [60], il existe un ordonnancement optimal avec au plus trois transports.*

4.4 Schéma d'approximation complètement polynomial ($FPTAS$)

Comme le problème considéré est NP-difficile, il est intéressant de développer des algorithmes polynomiaux qui donnent avec une certaine garantie des solutions proches de la solution optimale. Ce qui nous conduit à la recherche des algorithmes d'approximation ou des schémas d'approximation. Le but de ces schémas d'approximation consiste à obtenir des solutions réalisables en temps polynomial, de valeur proche de la valeur optimale avec un rapport de performance fixé à l'avance. Les différentes définitions et le principe de ces algorithmes sont donnés dans la section 1.6.3. Ce procédé a été appliqué pour la résolution des problèmes d'ordonnancemnt, nous citons quelques résultats :
Un schéma d'approximation polynomial ($PTAS$) a été proposé pour le problème NP-difficile $P2||C_{max}$ [74], l'idée est de classer les tâches en deux catégories : les petites tâches et les grandes tâches. Cette classification dépend du paramètre ε, tel que $0 < \varepsilon < 1$. Une tâche est considérée comme une grande tâche si sa durée opératoire est supérieure à εL où L est une borne inférieure $(L = \max\left\{P_{max}, \frac{1}{2}\sum_{j=1}^{n} p_j\right\})$.
Le problème à deux machines parallèles identiques avec contraintes de précédence sous forme de chaîne (*chain-precedence constraints*) a été traité par Agnetis et al. [1], un schéma d'approximation complètement poynomial en $O(\frac{n^4}{\varepsilon})$ a été proposé. Une autre application est donnée par

Kovalyov [59] pour un problème d'ordonnancement sur machine unique avec date échue pour la minimisation de la tardivité totale. La modification adoptée porte sur les durées opératoires ainsi que sur les dates échues. Ji et Cheng [56] ainsi que Woeginger [85] ont proposé des schémas d'approximation pour le problème sur machines parallèles avec des contraintes de type *"a grade of service provision"*. Un schéma d'approximation polynomial ($PTAS$) de complexité $O(m^{f(\varepsilon)}+n)$ a été proposé par Fishkin et al. [42] pour le problème avec préemption et délais de transpport $P|pmtn(delay_{ii'}=d)|C_{max}$.

Rappelons que l'objectif principal lors de la construction d'un schéma d'approximation est de transformer une instance du problème considéré (qui est dite instance difficile ou instance originale) en une autre instance plus simplifiée. Par la suite, on résoud la nouvelle instance par une méthode exacte. Finalement, à partir de la solution optimale du problème simplifié, on détermine une solution approchée pour l'instance originale.

Considérons le problème $P2|pmtn(delay_{ii'})|C_{max}$. Rappelons que le problème est NP-difficile, un algorithme (DPX) en $O(n^2UB)$ est proposé pour sa résolution. Dans ce qui suit, nous proposons un schéma d'approximation complètement polynomial ($FPTAS$) pour ce problème.

Soit UB une borne supérieure, $UB=\sum_{j=1}^{n} p_j$, cette borne supérieure est identique à celle proposée dans [22].

Soit LB une borne inférieure du C_{max}, $LB=max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}$, dans le cas de deux machine $LB=max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{2}\sum_{j=1}^{n} p_j\right\}$.

Une nouvelle instance du problème est définie comme suit :
Soit $\varepsilon>0$, $K=\frac{\varepsilon LB}{n}$.
Pour chaque tâche T_j, $\forall\ j=\overline{1,n}$, le nouveau temps de traitement est défini par $p'_j=\lfloor\frac{p_j}{K}\rfloor$.
Avec cette transformation, nous appliquons l'algorithme DPX sur l'instance simplifiée. La solution de l'instance originale est obtenue à partir de la solution de l'instance simplifiée.
Ces différentes étapes constituent l'algorithme suivant :

Algorithme 7 : *FPTAS*

Début

pour $j := 1$ ***à*** n **faire**

| • $p'_j = \lfloor \frac{p_j}{K} \rfloor$;

fin

• Résoudre le problème modifié $P2|pmtn(delay_{ii'})|C_{max}$ par l'algorithme DPX ;

• Construire la solution du problème original à partir de la solution du problème modifié ;

Fin.

Nous proposons le théorème :

***Théorème* 4.4.1.** *[49] Le problème $P2|pmtn(delay_{ii'})|C_{max}$ admet un $FPTAS$ en $O(\frac{n^3}{\varepsilon})$.*

Preuve.

Soit $K = \frac{\varepsilon LB}{n}$ *(1)*

Les nouveaux temps de traitement sont $p'_j = \lfloor \frac{p_j}{K} \rfloor$, $\forall j = \overline{1,n}$.

Soit $\mathcal{P}$ le problème original et $\mathcal{P}'$ le problème simplifié (avec les instances modifiées par la transformation des temps de traitement).

Soit π_A la séquence des tâches, $C'_{max}(\pi_A)$ la valeur du C_{max} calculée par l'algorithme DPX pour résoudre le problème $\mathcal{P}'$. Ainsi, $c'_j(\pi_A)$ est la date de fin de traitement de la tâche T_j ordonnancée selon la séquence π_A.

Pour le problème $\mathcal{P}$, $c_j(\pi)$ correspond à la date de fin de traitement de la tâche T_j (avec le temps de traitement p_j), définie par la solution π avec $C_{max}(\pi)$. Soit π^* la solution optimale calculée par l'algorithme DPX pour le problème $\mathcal{P}$.

Par définition :

$$C'_{max}(\pi_A) \leq C'_{max}(\pi^*) \quad (2)$$

$$C_{max}(\pi_A) \geq C_{max}(\pi^*) \quad (3)$$

Pour chaque tâche T_j, on a :

$Kp'_j \leq p_j$ *(4)*

$p_j \leq K(p'_j + 1)$ *(5)*

(4) $\Rightarrow K \sum_{j=1}^{n} p'_j \leq \sum_{j=1}^{n} p_j$
Sachant qu'il n'y a pas de temps mort entre deux tâches de la séquence π *:*

$Kc'_j(\pi) \leq c_j(\pi) \Rightarrow KC'_{max}(\pi) \leq C_{max}(\pi)$ *(6)*

(5) $\Rightarrow \sum_{j=1}^{n}(p_j) \leq K \sum_{j=1}^{n}(p'_j + 1) \Rightarrow \sum_{j=1}^{n}(p_j) \leq K \sum_{j=1}^{n}(p'_j) + Kn$

$\Rightarrow c_j(\pi) \leq Kn + Kc'_j(\pi) \Rightarrow C_{max}(\pi) \leq Kn + KC'_{max}(\pi)$

Pour la séquence π_A*, on a :* $C_{max}(\pi_A) \leq KC'_{max}(\pi_A) + Kn$

(2) $\Rightarrow C_{max}(\pi_A) \leq KC'_{max}(\pi^*) + Kn$

(6) $\Rightarrow C_{max}(\pi_A) \leq C_{max}(\pi^*) + Kn$

(1) $\Rightarrow nK = \varepsilon LB$

$C_{max}(\pi_A) \leq C_{max}(\pi^*) + \varepsilon LB$

$\Rightarrow C_{max}(\pi_A) \leq (1 + \varepsilon)C_{max}(\pi^*)$. □

Le temps d'exécution de l'algorithme DPX est $O(n^2UB') = O(n^2\frac{UB}{K}) = O(n^2.\frac{UB.n}{\varepsilon LB}) = O(\frac{n^3}{\varepsilon}.\frac{UB}{LB}) = O(\frac{n^3}{\varepsilon})$.
D'où la complexité du schéma d'approximation $FPTAS$ proposé est en $O(\frac{n^3}{\varepsilon})$.

4.5 Expérimentations numériques

Pour évaluer les algorithmes DPX et $FPTAS$. Nous avons effectué des expérimentations en générant aléatoirement plusieurs instances. Le nombre de tâches n est pris dans $\{50, 100, 200,$ 300. Le nombre de machines est égal à 2 et pour chaque valeur de n nous avons généré 50 instances.

Deux types d'instances ont été utilisées pour les tests :

– Type 1 : Les mêmes paramètres que celle des expérimentations réalisées dans [22] ont été considérés. Les temps de traitement des tâches et les délais de transport sont générés de façon aléatoire suivant une loi uniforme dans les intervalles [1,30] et [1,100] respectivement. Les résultats de ce type d'instances sont résumés dans le tableau 4.3.
– Type 2 : Pour générer des instances difficiles (pour lesquelles l'algorithme de Mc Naughton ne donne pas une solution optimale), les temps de traitement sont générés en utilisant une loi uniforme sur l'intervalle [1,100], et les délais de transport sont pris dans l'intervalle $[LB - p_{min}, C_{max}^{LPT} + p_{min}]$. LB est la borne inférieure donnée par $LB = max\left\{\frac{1}{2}\sum_{j=1}^{n} p_j, p_{max}\right\}$, $p_{max} = \max_{1\leq j\leq n}\{p_j\}$ et $p_{min} = \min_{1\leq j\leq n}\{p_j\}$ (les tâches sont rangées dans l'ordre LPT). Les résultats obtenus sont résumés dans le tableau 4.4.

L'algorithme $FPTAS$ a été testé avec plusieurs valeurs de ε, nous avons reporté seulement les valeurs significatives à savoir $\varepsilon \in \{5\%, 10\%\}$.

Une étude comparative avec l'heuristique $H2V2$ proposée [22] a été faite. Le principe de cette heuristique est donnée à la fin de la section 2.6.

Pour chaque algorithme, nous avons calculé le temps d'exécution moyen (en secondes) et la déviation moyen par rapport à la solution optimale fournie par l'algorithme DPX. Cette déviation est définie par : $R_{FPTAS} = \frac{C_{max}^{FPTAS} - C_{max}^{DPX}}{C_{max}^{DPX}} \times 100$ et $R_{H2V2} = \frac{C_{max}^{H2V2} - C_{max}^{DPX}}{C_{max}^{DPX}} \times 100$. Nous avons reporté aussi la déviation maximale de $FPTAS$ et $H2v2$.

TAB. 4.3 – Résultats des instances de type 1

n	Temps d'exécution moyen (s)			Déviation moyenne			Déviation maximale		
	DPX	$FPTAS$		$FPTAS$		$H2V2$	$FPTAS$		$H2V2$
		$\varepsilon = 5\%$	$\varepsilon = 10\%$	$\varepsilon = 5\%$	$\varepsilon = 10\%$		$\varepsilon = 5\%$	$\varepsilon = 10\%$	
50	< 1	< 1	< 1	0.076	0.188	0.3109	1.048	1.3	1.193
100	< 1	< 1	< 1	0.015	0.072	0.125	0.4	0.51	0.259
200	< 1	< 1	< 1	0.0006	0.038	0.0516	0.066	0.18	0.099
300	< 1	< 1	< 1	0.0036	0.021	0.035	0.04	0.13	0.0831

D'après le tableau 4.3, qui correspond aux résultats des instances de type 1, le temps d'exécution moyen pour les instances avec 300 tâches est moins de 1 seconde pour les algorithmes DPX,

$FPTAS$ et $H2V2$. La déviation moyenne de $FPTAS$ est inférieure à 1% pour toutes les instances avec $\varepsilon \in \{5\%, 10\%\}$. En ce qui concerne l'heuristique $H2V2$, la déviation moyenne est d'environ 0,31 % pour les cas de 50 tâches et la déviation maximale est d'environ 1,2 %.

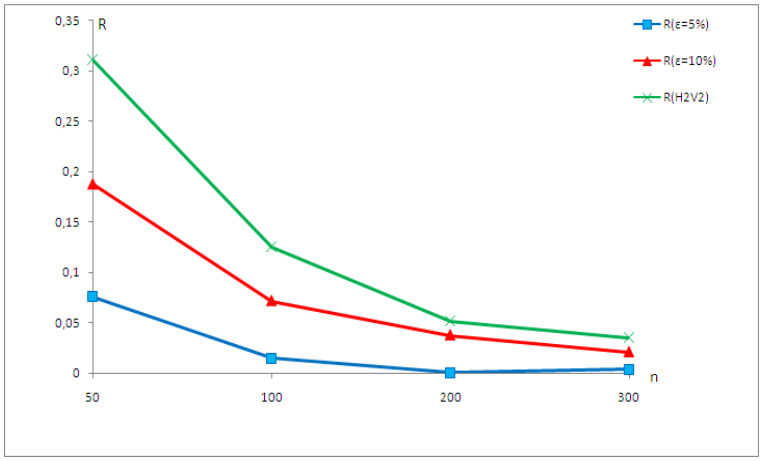

FIG. 4.1 – Déviations moyennes des instances de type 1

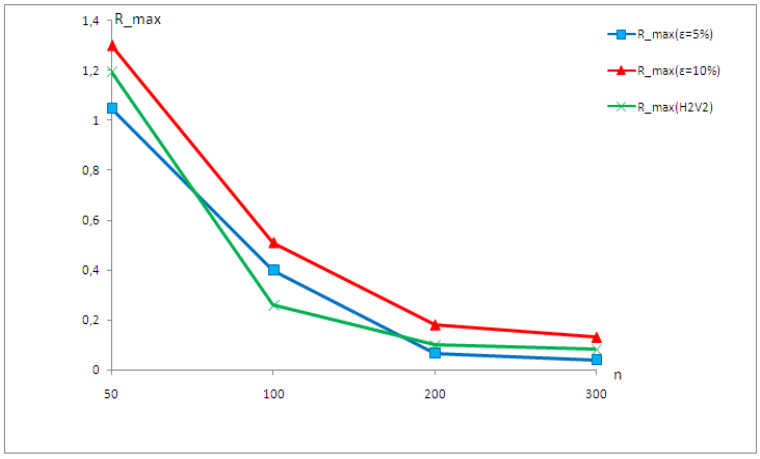

FIG. 4.2 – Déviations maximales des instances de type 1

De la représentation graphique (figure 4.1), il est claire que la déviation moyenne diminue lorsque le nombre de tâches augmente. Ces résultats sont proches de ceux obtenus dans [22].

Les résultats des instances de types 2 (instances difficiles) sont représentés dans le tableau 4.4.

TAB. 4.4 – Résultats des instances de type 2

n	Temps d'exécution moyen (s)			Déviation moyenne			Déviation maximale		
	DPX	$FPTAS$		$FPTAS$		$H2V2$	$FPTAS$		$H2V2$
		$\varepsilon = 5\%$	$\varepsilon = 10\%$	$\varepsilon = 5\%$	$\varepsilon = 10\%$		$\varepsilon = 5\%$	$\varepsilon = 10\%$	
50	0.36	0.26	0.18	0.15	0.173	5.39	0.54	0.61	6.46
100	2.68	2.11	1.46	0.096	0.089	2.83	0.42	0.42	3.25
200	22.25	20.45	12.33	0.022	0.021	1.39	0.101	0.101	1.53
300	75.47	58.18	41.23	0.017	0.017	0.93	0.056	0.056	0.97

D'après le tableau 4.4, le temps d'exécution moyen de l'algorithme DPX pour les instances avec 300 tâches est d'environ 76 secondes tandis que le temps d'exécution moyen de l'algorithme $FPTAS$ est inférieur à 60 secondes pour $\varepsilon = 5\%$ et moins de 42 secondes pour $\varepsilon = 10\%$.

Selon la figure 4.3, le temps moyen d'exécution augmente rapidement avec la taille des instances.

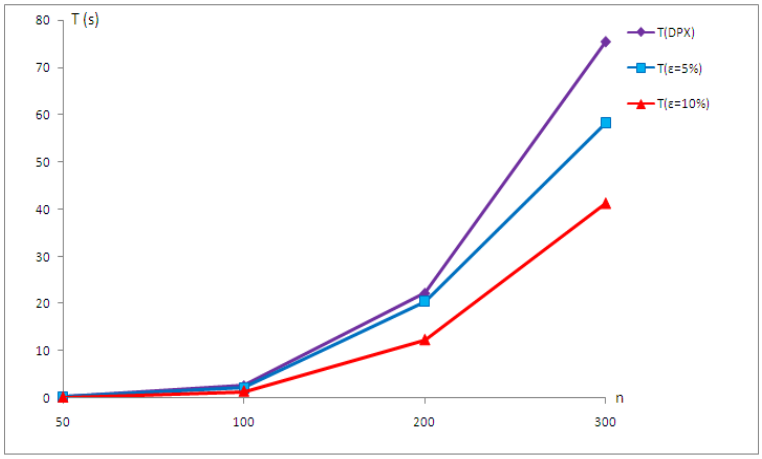

FIG. 4.3 – Temps d'exécution moyen des instances de type 2

La déviation moyenne de la solution donnée par $FPTAS$ par rapport à la solution optimale reste inférieure à 1 % pour toutes les instances avec $\varepsilon \in \{5\%, 10\%\}$ et diminue lorsque le

nombre de tâches augmente. Concernant l'heuristique $H2V2$, elle donne des solutions qui sont, en moyenne, à 5,4% de la solution optimale pour les instances avec 50 tâches, et avec un écart maximal d'environ 6,5%. Ces résultats sont très différents comparant avec les résultats obtenues sur les instances de type 1. Ceci est expliqué par le fait d'avoir des délais importants, il y aura des cas où un décalage est effectué pour respecter les délais de transport, donc un temps mort sera inséré et ceci fait augmenter la valeur du C_{max}. Notons aussi que toutes les solutions retournées par $H2V2$ sont dominées par celles calculées par $FPTAS$ qui donne une solution optimale dans la majorité des cas.

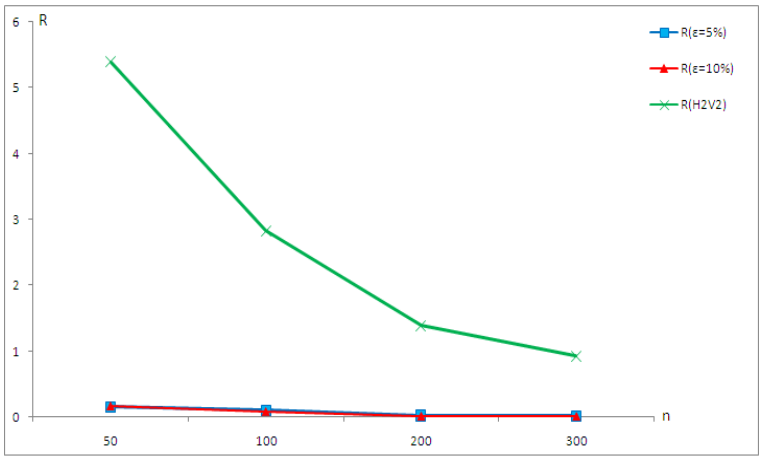

FIG. 4.4 – Déviations moyennes des instances de type 2

Contrairement au temps d'exécution moyen qui est proportionnel à la taille des instances, la déviation moyenne ainsi que la déviation maximale sont inversement proportionnelles à la taille des instances (figures 4.4 et 4.5). On voit clairement que la déviation moyenne et la déviation maximale de l'heuristique $H2V2$ diminuent rapidement lorsque la taille des instances augmente. La déviation moyenne est égale à 5.39% pour les instances avec 50 tâches, cette déviation diminue progressivement jusqu'à atteindre 0.97% pour les instances avec 300 tâches. De même pour la déviation maximale, elle passe de 6.46% pour les instances avec 50 tâches à 0.97% pour les instances avec 300 tâches.

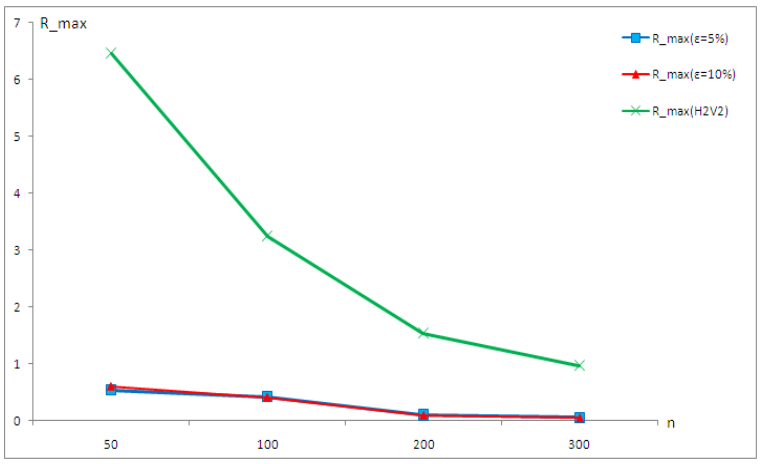

FIG. 4.5 – Déviations maximales des instances de type 2

4.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés particulièrement au problème à deux machines. Une méthode de résolution exacte de type programmation dynamique a été proposée. Nous avons proposé aussi un schéma d'approximation complètement polynomial.

Les expérimentations numériques ont montré la performance des méthodes proposées pour la résolution du problème même avec des instances de grande taille.

Chapitre 5

Résolution du problème général

5.1 Introduction

Dans ce qui suit, nous considérons le problème général $Pm|pmtn(delay_{ii'})|C_{max}$. Rappelons que ce problème est NP-difficile, puisque le problème qui se réduit à deux machines et délais de transport identiques est NP-difficile. Pour ce dernier, une méthode exacte de type programmation dynamique a été proposée. Afin de résoudre le problème général avec m machines, nous proposons une stratégie de résolution qui comporte deux phases [48] :

– La première phase consiste à trouver la séquence des machines, c'est-à-dire un ordre pour les machines de telle sorte à ce que les délais de transport soient au minimum.

– La deuxième phase de la résolution consiste à affecter les tâches aux machines en utilisant des heuristiques. Ces heuristiques font que les solutions obtenues aient au plus $(m-1)$ tâches préemptées.

5.2 Détermination de la séquence des machines

Pour déterminer la meilleure séquence, nous proposons de modéliser le problème par un graphe, tel que l'ensemble des sommets correspond à l'ensemble des machines. Nous faisons correspondre

un sommet à chaque machine, les arcs sont pondérés par les délais de transport. Ainsi le graphe obtenu par la modélisation proposée est un graphe complet pondéré. La détermination de la meilleure séquence des machines revient à déterminer un chemin hamiltonien de longueur minimum (voir definition 1.7.10).
Pour bien voir la modélisation du problème, on propose l'exemple suivant :

Exemple 5.2.1. *Soit le problème avec 5 machines, les délais de transport sont donnés par le tableau 5.1. Le graphe aura 5 sommets, chaque sommet correspond à une machines. Les arcs représentent les liens entre les machines, chaque ars est pondéré par le délai de transport nécessaire pour passer d'une machine M_i à une autre machine $M_{i'}$.*

TAB. 5.1 – Délais de transport

	M_1	M_2	M_3	M_4	M_5
M_1	0	24	1	29	5
M_2	29	0	10	3	14
M_3	5	9	0	16	9
M_4	17	3	18	0	18
M_5	18	21	3	12	0

Le chemin passant par les sommets $M_1 \overset{1}{\rightarrow} M_3 \overset{9}{\rightarrow} M_2 \overset{3}{\rightarrow} M_4 \overset{18}{\rightarrow} M_5$ est un chemin hamiltonien de longueur égale à 31. Un autre chemin passant par les sommets $M_1 \overset{5}{\rightarrow} M_5 \overset{3}{\rightarrow} M_3 \overset{9}{\rightarrow} M_2 \overset{3}{\rightarrow} M_4$ est de longueur égale à 20.

Le problème du chemin hamiltonien est aussi difficile que celui du circuit hamiltonien. Ce dernier correspond au problème du voyageur de commerce. Plusieurs méthodes de résolution ont été proposées. Parmi ces méthodes les métaheuristques, nous allons adopter l'une d'entre elles : algorithme de colonies de fourmis. Le choix de l'adaptation de cette métaheuristique se justifie par l'analogie du problème de recherche du plus court chemin avec le principe de la métaheuristique.

5.2.1 Algorithme de colonies de fourmis

Les algorithmes de colonies de fourmis constituent une métaheuristique inspirée du comportement des insectes. Le premier algorithme de colonies de fourmis *Ant System (AS)* [36] proposé par Dorigo, Colorni et Maniezzo en 1991, vise à résoudre le problème du voyageur de commerce. Nous avons introduit le mécanisme de cette métaheuristique au premier chapitre, dans la section réservée aux méthodes de résolution (section 1.6). Nous donnons le principe de cet algorithme dans ce qui suit.

L'algorithme de colonie de fourmis repose sur le principe suivant :

Durant chaque itération, m fourmis construisent un tour en exécutant n étapes. Une règle de décision probabiliste est appliquée pour séléctionner à chaque fois le nœud à visiter. En effet, étant sur un nœud i la fourmi choisit un nœud j et rajoute l'arc (i, j) au chemin. Cette étape est répétée jusqu'à ce que la fourmi complete sa tournée.

Dès que les fourmis finissent leurs tours, chacune dépose une quantité de phéromone qui permet de garder la trace des arcs visités afin de les favorisés pour les fourmis suivantes. La quantité de phéromone associée à l'arc (i, j) à l'instant t est notée $\tau_{ij}(t)$, elle indique la profitabilité d'emprunter l'arc (i, j). Cette quantité est inversement proportionnelle à la qualité du parcours c'est-à-dire plus le chemin est court plus la quantité est grande.

À chaque itération, une partie de la phéromone s'évapore pour éviter les accumulations de la trace à l'infini. Ainsi, les arcs dont la trace n'est plus renouvelée deviennent moins attirants.

Une liste tabou est associée à chaque fourmi pour assurer les chemins hamiltoniens. Ces listes sont vidées à chaque itération après la mise à jour de la trace de phéromone.

Chaque fourmi doit respecter un certain nombre de règles, parmi lesquelles :

- Une fourmi k ne peut visiter qu'une fois chaque ville, d'où l'utilité des listes tabou.
- Le choix d'une ville est établi à l'aide d'une probabilité de transition calculée en fonction de la visibilité η_{ij} qui est inversement proportionnelle à la distance d_{ij} entre les deux villes i et j, $\eta_{ij} = \frac{1}{d_{ij}}$.
- La probabilité de transition d'une fourmi k, notée P_{ij}^k appelée règle de transition proportionnelle aléatoire [38]. P_{ij}^k se calcule par l'équation suivante :

$$\mathrm{P}_{ij}^{k} = \begin{cases} \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{w \notin tabou_k}[\tau_{iw}]^{\alpha}[\eta_{iw}]^{\beta}}, & si \;\; j \notin tabou_k; \\ 0 & sinon. \end{cases} \quad (5.1)$$

Rappelons que τ_{ij} est la quantité de phéromone déposée sur l'arc (i, j)

α, β sont deux paramètres qui contôlent l'importance relative de l'intensité de la trace et la visibilité.

- Une fois la ville choisie, elle sera rajoutée à la liste tabou de la fourmi k. Cette liste contient, dans l'ordre, les villes visitées par la fourmi k.

- Chaque fourmi dépose une quantité de phéromone proportionnelle à la qualité du trajet une fois qu'il soit terminé. Cette quantité est notée $\Delta\tau_{ij}$ calculée selon l'équation :

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k}, & si\ l'arc\ (i,j)\ est\ \ dans\ \ le\ trajet\ de\ la\ fourmi\ k; \\ 0, & sinon. \end{cases} \quad (5.2)$$

L_k est la longueur du trajet effectué par la fourmi k.

- Une certaine quantité de la phéromone déposée va s'évaporer à chaque itération selon la règle suivante :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \quad (5.3)$$

$\rho\tau_{ij}(t)$ est la quantité de phéromone évaporée. $\rho \in [0, 1]$ représente le taux d'évaporation de la phéromone (*the pheromone evaporation rate*).

Les différents paramètres de l'algorithme sont définis par :

- La quantité de phéromone initiale déposée sur chaque arc τ_0. Plusieurs valeurs sont proposées pour la valeur initiale τ_0 selon la version de l'algorithme (voir [38]).

  La trace de phéromone est sauvegardée dans une matrice Γ qui a la même structure que la matrice d'adjacence du graphe.
- Le nombre maximun d'itération NI_{max} qui détermine quand faudra-il arrêter l'algorithme.
- Le nombre de fourmis m est fixé expérimentalement. Les résultats montrent qu'il peut être égal au nombre de villes pour les différentes variantes de l'algorithme [38].
- Les deux paramètres positifs α et β, indiquent la relation entre la visibilité et la trace de phéromone déposée. Dorigo et al. ont constaté que α devrait être autour de 1 et β est compris entre 2 et 5 [38].
- Le paramètre $\rho \in [0, 1]$ représente le taux d'évaporation de la trace de phéromone.

Un tableau récaputilatif donnant les valeurs des différents paramètres selon la version de l'algorithme est présenté dans [38]. Pour plus de détails concernant cette métaheuristique sont donnée dans [17, 37, 38] et [84].

L'algorithme OCF résume les différentes étapes de cette métaheuristique :

Algorithme 8 : OCF

Début

- Initialiser la trace de phéromone de tous les arcs à τ_0 ;
- $iter := 1$ ($iter$ donne le nombre d'itérations) ;

répéter

- Répartir les m fourmis aléatoirement sur les villes ;

pour chaque $k := 1$ ***à*** m **faire**

- Poser $S = \{1, 2, \ldots, m\}$ (l'ensemble des villes) ;
- Chaque fourmi k place sa ville de départ (i) choisie aléatoirement dans sa liste tabou notée $tabou_k$;

répéter

- Chaque fourmi k, choisit la prochaine ville à visiter selon la probabilité de transition P_{ij}^k calculée par l'équation (5.1) ;
- Poser $S = S \setminus \{j\}$; $tabou_k := tabou_k \cup \{j\}$; $i := j$;

jusqu'à $S = \emptyset$;

fin

- Une fois que les fourmis finissent la construction de leurs tours, elles déposent sur chaque arc (i, j) du trajet une quantité de phéromone $\Delta\tau_{ij}$ calculée par l'équation (5.2) ;
- La quantité de phéromone est mise à jour selon l'équation (5.3) ;
- $iter := iter + 1$;

jusqu'à $iter = NI_{max}$;

Fin.

Adaptation des algorithmes de colonies de fourmis aux problèmes d'ordonnancement

Initialement, les algorithmes de colonies de fourmis ont été conçus pour la résolution du problème de voyagueur de commerce qui a été considéré comme un problème test. Ce problème est NP-difficile, il consiste à chercher le plus court chemin hamiltonien dans un graphe (voir définition 1.7.10), d'où l'approche avec les colonies de fourmis.
Depuis leur introduction, les algorithmes de colonies de fourmis ont connu plusieurs variations, des recherches ont été menèes dans deux directions. Une pour le développement des algorithmes et l'amélioration de leurs performance par l'étude de leur comportement suite aux variations des paramètres. Plusieurs résultats ont été publiés dans ce contexte [38, 37] et [17]. L'autre direction consiste à appliquer ces algorithmes pour la résolution des problèmes d'optimisation combinatoire. Les premières applications étaient destinées à la résolution du problème du voyageur de commerce [35]. Au fil du temps, ces applications se sont élargies pour toucher plusieurs poblèmes d'optimisation combinatoire, par exemple le problème de tournées de véhicules [9]. Ce problème consiste à déterminer les tournées d'une flotte de véhicules qui assurent la livraison d'un certain nombre de clients, ou de réaliser des tournées pour le transport des personnes (ramassage scolaire ou transport du personnel d'une entreprise par exemple).
Nous allons nous intéresser particulierement aux applications des algorithmes de colonies de fourmis pour la résolution des problèmes d'ordonnancement. Nous citons quelques travaux tel que [67] et Solnon [82] concernant le problème d'ordonnancement des véhicules (*car-sequencing problem*). Les problèmes d'ordonnancement dans les différents types d'ateliers ont fait aussi l'objet de ces applications. Dans [88], Yagmahan et Yenisey ont proposé un problème d'ordonnancement de type flow-shop multi-objectif a été abordé. Le cas du flow-shop pour la minimisation d'un seul objectif (C_{max}) a été traité dans [81]. Huang et Liao [54] ont traité le cas d'un atelier de type job-shop. Enfin le cas des machines parallèles a été traité par Behnamian et al. [8] avec des temps de préparation dépendant de la séquence. Dans [64], une adaptation de l'algorithme de colonies de fourmis a été proposée pour la résolution du problème d'ordonnancement avec contrainte de précédence $P|prec|C_{max}$.

Nous avons montré que notre problème avec des durées opératoires identiques, $P|pmtn\ (delay_{ii'})$,

$p_j = p|C_{max}$, est NP-difficile (théorème 3.4.1). La preuve est basée sur une réduction au problème de la chaîne hamiltonienne (pour l'instance considérée les délais sont constants). De là, nous constatons la similitude de ce dernier avec le problème de la recheche de la meilleure séquence des machines. En effet, la recherche de la meilleure séquence consiste à déterminer l'ordre dans lequel les différents transports vont être effectués. Ces transport se font à l'aide d'un transporteur qui passe une seule fois par chaque machine afin de prendre et/ou déposer une tâche préemptée. Nous remarquons que le problème revient à résoudre le problème de la recherche du plus court chemin hamiltonien (dans le cas général, les délais de transport sont quelconques).

Pour appliquer l'algorithme de fourmis au problème de recherche de la séquence des machines, nous avous remplacer la matrice des distances par celles des délais de transport. Le calcul de la probabilité de transition se fait alors en fonction des délais qui interviennet dans la visibilité η_{ij} tel que $\eta_{ij} = \frac{1}{delay_{ij}}$. Le nombre de fourmis sera fixé au nombre de machines, les autres paramètres seron fixés aux valeurs indiquées à la section relative aux expérimentations.

À la sortie de l'algorithme, la liste des machines obtenue correspond aux sommets du chemin hamiltonien de longueur minimum. Pour déterminer la séquence des machines, il suffit de prendre les machines de la liste dans l'ordre inverse. La raison pour laquelle l'ordre des machines est inversé sera expliquée lors de l'exposition des heuristique d'affectation dans la sous section suivante.

5.2.2 Heuristique H_TR

Nous proposons également une heuristique H_TR permettant de déterminer la séquence des machines. L'heuristique H_TR est inspirée de l'heuristique du plus proche voisin conçue pour la résolution du problème du voyageur de commerce. Cette heuristique consiste à chercher un tour à partir d'un sommet arbitraire. À chaque itération, elle cherche un sommet non encore visité qui soit le plus proche du dernier sommet visité.
L'heuristique H_TR que nous avons proposé dans [48] consiste à chercher le meilleur chemin commençant par une machine M_i ($\forall i = \overline{1, m}$). À la fin le chemin k de longueur D_k vérifiant

$D_k = \min_{1 \leq i \leq m} \{D_i\}$ est choisi parmi les m chemins, de longueurs D_i, obtenus.
Avant de donner les différentes étape de cette heuristique, nous donnons quelques notations :
Soit M l'ensemble des machines $|M| = m$, S l'ensemble des machines constituant le chemin et D_i la longueur du chemin commençant par la machine M_i.
Cette heuristique comporte les étapes suivantes :

Algorithme 9 : H_TR

Début

pour $i = 1$ *à* m **faire**

- Initialisation $S := \emptyset$;$L := 0$;
- Poser $S := S \cup \{i\}$; $M := M \setminus \{i\}$;

répéter

- Chercher la machine la plus proche de la machine M_i (soit M_k cette machine) ;
- $S := S \cup \{k\}$; $M := M \setminus \{k\}$;
- $L := L + delay_{ik}$;
- $i := k$;

jusqu'à $M = \emptyset$;

- $D_i := L$;

fin

- Chercher $\min_{1 \leq i \leq m} \{D_i\}$;

Fin

Cette heuristique détermine le plus court chemin en $O(m^2)$.

En fait, cette heuristique va être utilisée plus tard lors des expérimentations pour comparer les heuristiques d'affectation entre elles.

5.3 Heuristiques d'affectation

Les heuristiques d'affectation, que nous proposons, se basent sur le principe des algorithmes de liste. La liste initiale correspond à la liste obtenue suivant la règle LPT (*Longest Prossing*

Time). Rappelons que cette règle consiste à ranger les tâches dans l'ordre décroissant de leur durée opératoire (temps de traitement). L'affectation des tâches se fait en prenant les machines dans l'ordre inverse du chemin obtenu à la première phase de la résolution, c'est-à-dire, en commançant par remplir la dernière machine visitée puis celle qui la prècède et ainsi de suite. Considérons deux machines successives de la meilleure séquence obtenue par l'application de l'algorithme de fourmis ou par l'heuristique H_TR, soit M_i et M_{i+1} ces deux machines. Nous commençons par remplir la machine M_{i+1} en lui affectant les tâches de la liste puis la machine M_i. S'il y a une tâche préemptée, elle commence son traitement sur la machine M_i à la première position et se poursuit sur la machine M_{i+1} à la dernière position.

5.3.1 Heuristique 1 : $Aff\_tâches$

La première heuristique $Aff\_tâches$ [48] consiste à affecter les tâches rangées selon l'ordre LPT à une machine jusqu'à la borne inférieure $LB = max\left\{\max_{1\leq j\leq n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}$. Si on n'arrive pas à exécuter entièrement une tâche avant LB, alors elle va être préemptée. Si le délai est respecté on fait la préemption et le transport. Dans le cas contraire, on cherche une autre tâche parmi les tâches restantes et on vérifie si on peut la traiter entièrement, sinon on cherche une tâche pour laquelle le délai est respecté. Dans le cas ou on ne trouve pas de tâches qui vérifient l'une des deux conditions, on fait la préemption avec un décalage pour respecter le délai de transport. On compare à la fin la solution trouvée par cette heuristique et la solution sans préemption déterminée à l'aide de l'algorithme FAM défini dans la section 2.4.2.
Nous donnons quelques notations qui vont être utilisées dans la description de l'algorithme. Notons par N l'ensemble de toutes les tâches et NS l'ensemble des tâches non encore traitées. t définit la date de début de traitement.
Les étapes de cette heuristique sont données par l'algorithme suivant :

Algorithme 10 : *Aff_tâches*

Début

- Ranger les tâches selon la règle LPT ;
- Calculer $LB = \max \left\{ \max\limits_{1 \leq j \leq n} \{p_j\} ; \frac{1}{m} \sum\limits_{j=1}^{n} p_j \right\}$;
- Initialisation : $t = 0$; $NS := N$;

tant que *($NS \neq \emptyset$)* **faire**

(1) **si** *($t + p_j < LB$)* **alors**

- Traiter la tâche T_j sur la machine M_i à l'instant t ;
- $NS := NS \setminus \{T_j\}$; $t := t + p_j$; $j := j + 1$;

sinon

(2) **si** *($t + p_j = LB$)* **alors**

- Traiter la tâche T_j sur la machine M_i à l'instant t et la tâche suivante sur la machine suivante à l'instant $t = 0$;
- $NS := NS \setminus \{T_j\}$; $t := 0$; $j := j + 1$; $i := i + 1$;

sinon

(3) **si** *($p_j + delay_{ii+1} \leq LB$)* **alors**

- La tâche T_j est coupée en deux parts, elle est traitée à la dernière position sur la machine M_{i+1} pour $(LB - t)$ unitées de temps et à la première position sur la machine M_i ;
- $NS := NS \setminus \{T_j\}$;

sinon

- Chercher une tâche T_{ind} tel que $(t + p_{ind} \leq LB)^{(a)}$, aller à (1) ou (2) ;
- S'il n'existe pas une tâche qui verifie l'inégalité (a), chercher une tâche T_{ind} tel que $(p_{ind} + delay_{ii+1} \leq LB)$ et aller à (3) ;

Autrement, la tâche T_j sera préemptée. Un décalage, qui permet de changer la date de reprise de la tâche préemptée pour respecter le délai de transport, est effectué ;

fin

fin

fin

fin

Fin

Cette heuristique donne une solution en $O(nlogn)$ et elle vise à minimiser le nombre de tâches préemptées.

Exemple 5.3.1. *Considérons une instance du problème* $Pm|pmtn(delay_{ii'})|C_{max}$*, avec* $n = 10$ *tâches et* $m = 5$ *machines. Les temps de traitement et les délais de transport sont données dans les tableaux suivants.*

TAB. 5.2 – Temps de traitement

T_j	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_j	5	6	1	6	5	4	5	6	6	2

TAB. 5.3 – Délais de transport

	M_1	M_2	M_3	M_4	M_5
M_1	0	24	1	29	5
M_2	29	0	10	3	14
M_3	5	9	0	16	9
M_4	17	3	18	0	18
M_5	18	21	3	12	0

La séquence des machines est donnée par : $M_4 \overset{3}{\rightarrow} M_2 \overset{10}{\rightarrow} M_3 \overset{5}{\rightarrow} M_1 \overset{5}{\rightarrow} M_5$.
Donc le transport se fait de la machine M_4 à la machine M_2 et de la machine M_2 à la machine M_3 et ainsi de suite. S'il y a une tâche préemptée, elle commence son traitement sur la machine M_4 puis elle sera transportée vers la machine M_2 où elle sera traitée à la dernière position.

La solution donnée par l'heuristique $Aff\_tâches$ est représentée par la figure 5.1 (une croix indique l'instant de préemption). Nous remarquons qu'il y a deux tâches préemptées T_1 et T_6. La tâche T_1 est transportée de la machine M_4 à la machine M_2. Quant à la tâche T_6, elle est transportée de la machine M_3 à la machine M_1.

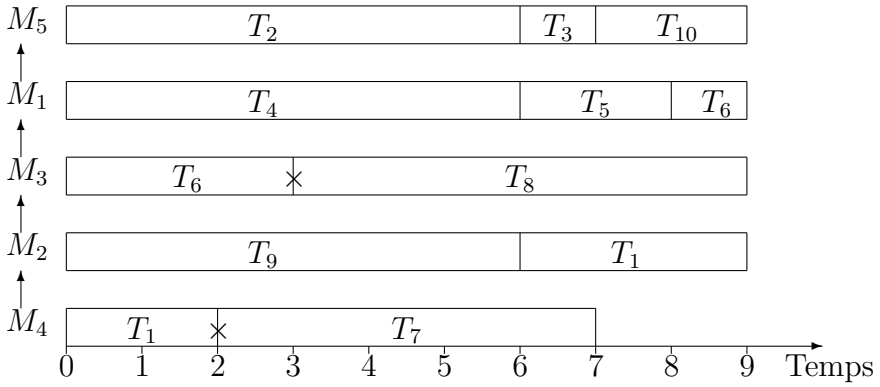

FIG. 5.1 – Solution donnée par l'heuristique $Aff\_tâches$

5.3.2 Heuristique 2 : H_List

Une autre heuristique H_List [48] consiste à générer des listes de tâches à partir d'une première liste qui correspond à la solution trouvée par l'heuristique $Aff\_tâches$. Ces listes sont obtenues par la permutation d'une tâche préemptée avec une autre tâche non préemptée choisie aléatoirement. À la fin au plus $(m-1)$ listes sont obtenues (car sous l'hypothèse de la validité de la conjecture posée dans [42], il y a au plus $(m-1)$ tâches préemptées, m est le nombre de machines). Après l'affectation des listes, nous choisissons la meilleure solution qui correspond à la meilleure liste.

Cette heuristique utilise deux procédures, la première procédure $Permut\_list$ est utilisée pour générer les différentes listes. La deuxième procédure Aff_list consiste à faire l'affectation des différentes listes générées.

Description de la procédure $Permut\_list$

Cette procédure opère de la manière suivante :

La liste initiale est obtenue à partir de la solution trouvée par l'heuristique $Aff\_tâches$ en prennant les tâches affectées à la première machines (selon la séquence des machines) puis les tâches de la deuxième machine et ainsi de suite. Les duplications des tâches préemptées sont supprimées tout en gardant une trace de leur indice. Ce dernier va nous aider par la suite pour faire la permutation avec une autre tâche choisie de façon aléatoire. Les différentes étapes de cette procédure sont données dans l'algorithme suivant [48] :

Procédure *Permut_list* ;

Début

• Déterminer NB le nombre de tâches préemptées dans la solution trouvée par l'heuristique $Aff\_tâches$;

pour *k :=1 à* NB **faire**

- • Choisir aléatoirement une tâches non préemptée, soit *ind* l'indice de cette tâche ;
- • Permuter la tâche *ind* avec la $k^{ième}$ tâche préemptée ;
- • Sauvegarder la nouvelle liste ;

fin

Fin.

Les instructions de la boucle sont répétées NB fois sachant que $NB \leq (m-1)$, donc cette procédure est en $O(m)$.

Pour illustrer le fonctionnement de cette procédure, nous l'appliquons aux données de l'exemple 5.3.1.

La liste initiale est $Liste_0 = \{T_2, T_3, T_{10}, T_4, T_5, T_6, T_6, T_8, T_9, T_1, T_1, T_7\}$. Nous remarquons que les tâches préemptées T_6, T_1 sont dupliquées, en supprimant ces duplications, nous obtenons à la liste $Liste_0 = \{T_2, T_3, T_{10}, T_4, T_5, T_6, T_8, T_9, T_1, T_7\}$. Comme il y a deux tâches préemptées dans la liste initiale, il y aura deux listes différentes. La première est obtenue en permutant la tâches T_6 avec une tâche non préemptée choisie aléatoirement (par exemple la tâche T_3), la deuxième liste est obtenue par la permutation de la tâche T_1 avec une tâche non préemptée (T_8 par exemple). Les deux nouvelles listes sont :

$Liste_1 = \{T_2, T_6, T_{10}, T_4, T_5, T_3, T_8, T_9, T_1, T_7\}$, $Liste_2 = \{T_2, T_3, T_{10}, T_4, T_5, T_6, T_1, T_9, T_8, T_7\}$.

Une fois que les différentes listes sont déterminées, nous procédons à leur affectation par la procédure Aff_list définie dans ce qui suit.

Description de la procédure Aff_list

Cette procédure permet d'affecter les tâches des listes générées en remplissant les machines une par une. Les étapes de cette procédure sont données dans l'algorithme suivant [48] :

Procédure Aff_list

Début

• Initialisation $t := 0$; $i := 1$; $j := 1$; $C_{max} := LB$;

pour chaque *tâche* j *d'une liste* k **faire**

si $(t + p_j < C_{max})$ **alors**

• Affecter la tâche j à la machine i à l'instant t ;

• $t := t + p_j$;

sinon

si $(t + p_j = C_{max})$ **alors**

• Affecter la tâche j à la machine i a l'instant t, la tâche suivante sera affectée à la machine suivante ;

sinon

la tâche j sera préemptée et transportée ;

fin

fin

fin

Fin.

Finalement, l'algorithme de l'heuristique H_List [48].

Algorithme 11 : H_List

Début

pour $k := 1$ *à* NB **faire**

• Générer les différentes listes par la procédure $Permut\_liste$;

• Affectation des listes par la procédure Aff_liste ;

fin

• Choisir la meilleure solution ;

Fin

Cette heuristique est en $O(n)$. En effet, la procédure qui génère les listes est en $O(m)$ et la procédure d'affectation se fait en $O(n)$.

Nous proposons une autre heuristique $H2\_mod$, elle permet de donner la meilleure solution entre la solution trouvée par l'application de l'heuristique $H2V2$ [22] (algorithme 3 section 2.6) avec la séquence des machines déterminée par l'heuristique H_TR définie précédement et la solution donnée par l'algorithme FAM (algorithme 2 section 2.4.2) qui permet de résoudre le problème sans préemption.

Algorithme 12 : $H2\_mod$

Début

- Ranger les machines par l'heurisique H_TR ;
- Ranger les tâches selon la règle LPT ;
- Affecter les tâches aux machines suivant l'algorithme de Mc Naughton et faire un décalage si nécessaire, la solution obtenue est $C_{max}(1)$;
- Résoudre le problème par l'algorithme FAM, la solution obtenue est $C_{max}(2)$;
- Choisir la meilleure solution $C^*_{max} = \min \{C_{max}(1), C_{max}(2)\}$;

Fin

5.4 Expérimentations numériques

Pour tester les heuristiques proposées, plusieurs instances sont générées aléatoirement. La taille des instances est définie par le nombre de tâches n qui prend une des valeurs $\{10, 20, 50, 100, 200\}$ et le nombre de machines m qui prend l'une des valeurs $\{5, 10, 15, 20\}$. Les durées opératoires des tâches et les délais de transport sont générés selon une loi uniforme dans les intervalles [1,30], [1,50] et [1,100] respectivement.

La séquence des machines est déterminée à l'aide de l'algorithme de colonie de fourmis, dont les paramètres sont fixés aux valeurs suivantes :

- La quantité initiale de phéromone $\tau_0 = 0.5$.
- Le nombre maximun d'itération $NI_{max} = 10$.
- Le nombre de fourmis est égale au nombre de machines.
- Les deux paramètres α et β sont fixés à 1 et 5 respectivement.
- Le paramètre ρ est fixé à 0.5.

Pour chaque valeur de m et n nous générons 100 instances. Chaque instance est résolue par l'algorithme OCF et l'heuristique $Aff\_tâches$ puis par l'algorithme OCF et l'heuristique H_List et en fin par l'heuristique $H2V2$. Nous appliquons par la suite, l'heuristique H_TR suivi de l'heuristique $H2\_mod$. Pour chaque instance résolue, nous reportons le temps moyen d'exécution (en secondes) noté T et la déviation moyen par rapport à la borne inférieure LB. Cette déviation est définie par : $R = \frac{C^H_{max} - LB}{LB} \times 100$.

Les résultats obtenus sont présentés dans les tableaux 5.4, 5.5. La première et la deuxième colonne de chaque tableau indiquent le nombre de machines et le nombre de tâches respectivement. La troisième colonne donne le temps moyen d'exécution de l'algorithme de fourmis combiné avec l'heuristique $Aff\_tâches$ et la quatrième colonne donne la déviation moyenne de cette heuristique. Les autres colonnes représentent les temps moyens d'execution et les déviations moyennes des heuristiques $H2V2$, $H2\_mod$ et H_List respectivement.
Le tableau 5.4 représente les résultats des instances avec $delay_{ii'} \in [1,\ 50]$.

TAB. 5.4 – Résultats des instances avec délais dans $[1, 50]$

m	n	$Aff\_tâches$		$H2V2$		$H2\_mod$		H_List	
		T	$R\%$	T	$R\%$	T	$R\%$	T	$R\%$
5	10	<0.001	17.09	<0.001	34.74	<0.001	6.68	<0.001	3.70
	20	<0.001	0.52	<0.001	1.57	<0.001	2.31	0.0011	<0.01
	50	0.0015	<0.01	<0.001	<0.01	<0.001	0.22	0.002	<0.01
10	20	0.007	19.53	<0.001	44.06	<0.001	5.5	0.0087	4.09
	50	0.008	<0.01	<0.001	<0.01	<0.001	1.51	0.0157	<0.01
	100	0.0081	<0.01	<0.001	<0.01	<0.001	0.307	0.0171	<0.01
15	50	0.032	<0.01	<0.001	4.49	<0.001	3.21	0.041	<0.01
	100	0.0327	<0.01	<0.001	<0.01	<0.001	0.69	0.047	<0.01
	200	0.033	<0.01	0.0013	<0.01	0.0018	0.066	0.062	<0.01
20	50	0.096	1.48	<0.001	21.70	<0.001	5.48	0.11	0.28
	100	0.0965	<0.01	<0.001	0.0126	<0.001	1.64	0.012	<0.01
	200	0.0966	<0.01	<0.001	<0.01	0.0015	0.27	0.138	<0.01

Le temps moyen d'exécution ainsi que la déviation moyenne de chaque heuristique sont représentés par les figures 5.2, 5.3 respectivement. Pour chaque valeur de m fixée, il y a trois ensembles

de rectangles correspondants aux valeurs de n. Pour chaque valeur de n fixée, il y a quatre rectangles chacun correspond à une heuristique.

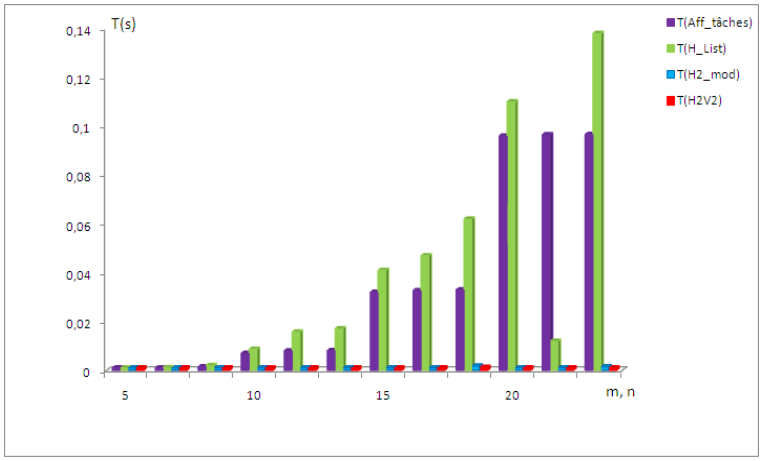

FIG. 5.2 – Temps d'exécution moyen, délais dans [1, 50]

Nous constatons que le temps moyen d'exécution des deux heuristiques $Aff\_tâches$ et H_List est considérable par rapport au temps moyen d'exécution des heuristiuqes $H2V2$ et $H2\_mod$. Ce temps est proportionnel à la taille des instances.

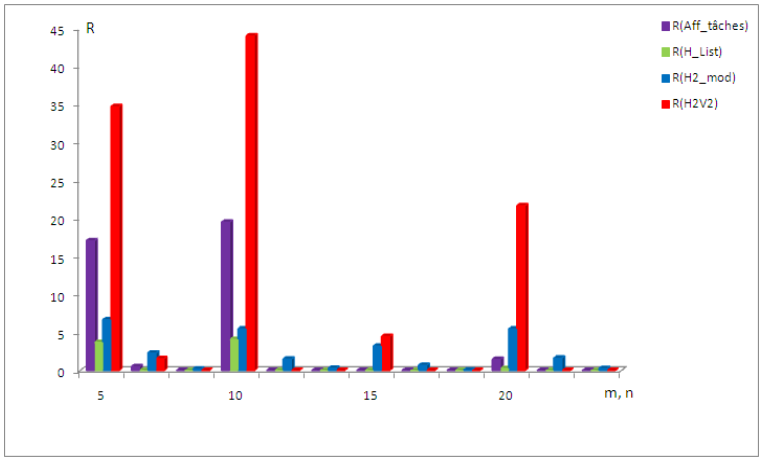

FIG. 5.3 – Déviations moyennes, délais dans [1, 50]

Nous remarquons aussi que la déviation moyenne de l'heuristique $H2V2$, sur les instances testées, est importante comparant au autres heuristiques (figure5.3).

Le tableau 5.5 représente les résultats des instances avec $delay_{ii'} \in [1,\ 100]$.

TAB. 5.5 – Résultats des instances avec délais dans $[1, 100]$

m	n	$Aff\_tâches$		$H2V2$		$H2\_mod$		H_List	
		T	$R\%$	T	$R\%$	T	$R\%$	T	$R\%$
5	10	0.0012	71.45	<0.001	84.81	<0.001	6.84	0.0015	36.31
	20	<0.001	12.54	<0.001	20.49	<0.001	2.58	0.0012	3.17
	50	<0.001	<0.01	<0.001	<0.01	<0.001	0.26	0.0067	<0.01
10	20	0.0075	76.97	<0.001	95.50	0.0011	6.97	0.011	33.062
	50	0.0102	0.78	<0.001	2.96	0.0012	1.6	0.025	<0.01
	100	0.0088	<0.01	<0.001	<0.01	<0.001	0.31	0.037	<0.01
15	50	0.032	21.28	<0.001	35.20	<0.001	3.20	0.0576	1.96
	100	0.033	<0.01	0.0015	0.45	0.0021	0.85	0.089	0.0895
	200	0.033	<0.01	0.0024	<0.01	0.0023	0.052	0.122	<0.01
20	50	0.098	49.60	0.0014	72.11	<0.001	5.73	0.147	13.59
	100	0.098	<0.01	<0.001	3.019	0.00125	1.48	0.178	<0.01
	200	0.098	<0.01	0.00157	<0.01	0.0023	0.26	0.208	<0.01

Ces tests nous mènent à faire les conclusions suivantes :

- Le temps moyen d'exécution des deux heuristiques $Aff\_tâches$ et H_List croît rapidement avec la taille des instances (figure 5.4). Le temps d'exécution de l'heuristique $Aff\_tâches$ et H_List est supérieur au temps d'exécution de l'heuristique $H2V2$ et $H2\_mod$ puisque la séquence des machines dans les deux heuristiques $Aff\_tâches$ et H_List est obtenue par l'appliquation de l'algorithme de colonies de fourmis dans la première phase de la résolution.

Le temps moyen d'exécution augmente avec le nombre de machines et de tâches. Nous remarquons qu'il est inférieur à 1 seconde dans presque tous les cas, que ça soit avec des délais dans $[1, 50]$ ou dans $[1, 100]$.

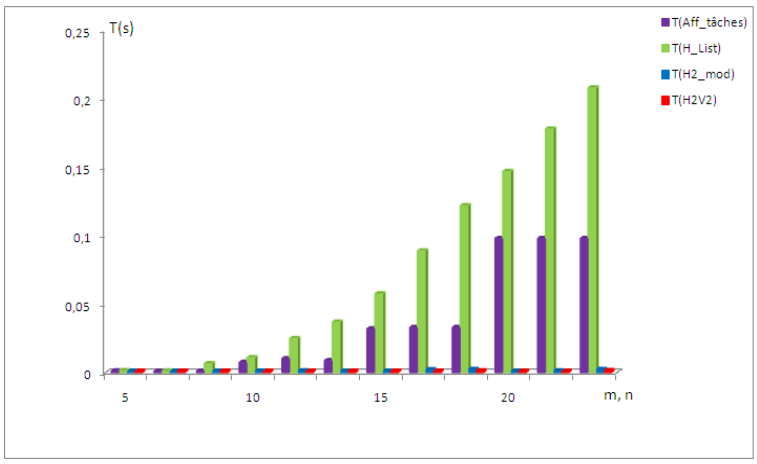

FIG. 5.4 – Temps d'exécution moyen, délais dans [1, 100]

- La déviation moyenne diminue pour les instances de grande taille. Lorsque les délais de transport sont générés dans l'intervalle $[1, 50]$, pour le cas de 10 tâches la déviation moyenne est d'environ $17,09\%$ et $34,74\%$ pour les heuristiques $Aff\_tâches$ et $H2V2$ respectivement.
En ce qui concerne les deux heuristiques $H2\_mod$ et H_List, leur déviation moyenne est meilleure, environ $6,68\%$ et $3,70\%$ respectivement. Pour le cas de 20 et 50 tâches, la déviation moyenne diminue de plus en plus. Cela peut s'expliquer par le fait qu'un grand nombre de tâches augmente l'écart entre les parties d'une tâche préemptée, puisque les délais de transport sont générés dans l'intervalle $[1, 50]$, ils seront respectés et la solution serai proche de la borne inférieure, dans ce cas l'algorithme de Mc Naughton donne la solution optimal. Ce résultat coïncide avec ceux trouvés dans [22].

- Les pires cas, sont obtenus pour les instances avec des délais de transport générés dans $[1, 100]$, car les délais sont importants par rapport aux durées opératoires des tâches, ce qui fait qu'un décalage est effectué dans certains cas pour respecter les délais de transport.

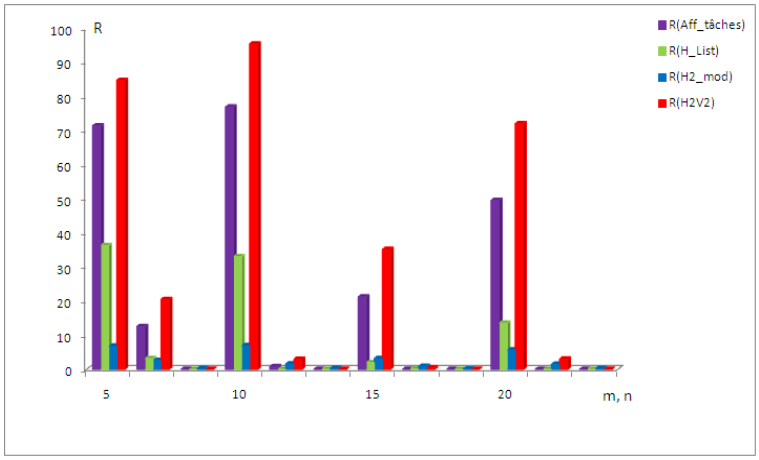

FIG. 5.5 – Déviations moyennes, délais dans [1, 100]

Pour ces instances, nous pouvons voir que la déviation moyenne de l'heuristique $H2\_mod$ est acceptable en comparant avec les autres heuristiques. Elle est près de 7% pour les instances avec 5 machines, 10 tâches et 10 machines, 20 tâches. Cette déviation diminue pour les instances de grande taille.

5.5 Conclusion

Le problème considéré dans ce chapitre est NP-difficile, la méthode proposée pour sa résolution repose sur deux phases. La première phase consiste à chercher la séquence des machines qui se ramène au problème du chemin hamiltonien. Ce problème est connu pour être NP-difficile, plusieurs méthodes existe pour sa résolution, nous avons opté pour un algorithme de colonies de fourmis. L'affectation des tâches constitue la deuxième phase, des heuristiques basées sur les listes de tâches sont proposées. Des expérimentations numériques ont été faites ainsi qu'une analyse comparative des différentes heuristiques proposées.

Conclusion

Le travail présenté dans cette thèse rentre dans le domaine de l'ordonnancement, particulièrement l'étude d'un problème d'ordonnancement sur machines parallèles identiques. Le but est la minimisation du temps de fin de traitement (*makespan*), en tenant compte des délais de transport des tâches préemptées.

En premier lieu nous avons donné quelques éléments de base concernant la terminologie liée à la complexité des algorithmes et aux problèmes d'optimisation combinatoire. Nous avons introduit aussi les différentes notions de la théorie de l'ordonnancement. Un exposé sur les méthodes de résolution et un bref rappel sur la théorie des graphes ont été présentés aussi.

Un état de l'art des principaux travaux traitant les problèmes d'ordonnancement sur machines parallèles avec de multiples contraintes a été donné. Des tableaux de synthèse récapitulant les principaux résultats ont été présentés dans le deuxième chapitre. Un intérêt particulier a été apporté aux problèmes d'ordonnancement avec contraintes portant sur les temps de préparation des tâches ou des machines, nous avons cité quelques travaux effectués dans ce sens. Notre plus grand intérêt c'est porté sur les problèmes d'ordonnancement avec délais de transport. Ces problèmes trouvent des applications pratiques dans le milieu industriel ou dans les systèmes informatiques.

Nous avons présenté un état de l'art sur les différents travaux traitant les problèmes d'ordonnancement avec contraintes sur les délais de transport. La fin du deuxième chapitre a été consacrée aux principaux résultats donnés pour le problème d'ordonnancent avec machines parallèles identiques sous contraintes de préemption et délais de transport.

Le traitement du problème posé s'est fait en trois parties :

- Dans la première partie, nous avons considéré un cas particulier du problème avec des tâches de même durée opératoire et avec des délais de transport identiques ou quelconques, les différents résultats ont fait l'objet du troisième chapitre.
- Dans la deuxième partie, qui constitue le quatrième chapitre, un programme dynamique et un schéma d'approximation complètement polynomial ont été proposés pour la résolution du problème avec deux machines.
- La troisième partie, a été dédiée à la résolution du problème général et ceci par une méthode en deux phases. La première phase de la résolution utilise une métaheuristique de type algorithme de colonies de fourmis afin de trouver la meilleure séquence des machines et par la suite déterminer l'ordre dans lequel vont s'effectuer les différents transports. La deuxième phase est consacrée à l'affectation des tâches, elle est basée sur des heuristiques d'affectation.

Dans la phase expérimentale de la deuxième et la troisième partie, nous avons généré plusieurs instances pour tester la performance des algorithmes proposés. Une analyse portant essentiellement sur l'observation des temps d'exécution des algorithmes et la déviation moyenne par rapport à une solution exacte (pour le problème à deux machines) et par rapport à une borne inférieure (pour le problème général) nous a conduits à tirer des conclusions sur l'efficacité des algorithmes proposés. Les résultats de ces expérimentations sont représentés à la fin du quatrième et du cinquième chapitre.

Nous résumons les principaux résultats dans le tableau suivant. La première partie de ce tableau récapitule les résultats lorsque le nombre de machines m n'est pas fixé et la deuxième partie représente les résultats pour un nombre fixé de machines.

Tableau récapitulatif

m	Problème	Resultats	Références
Non fixé	$delay_{ii'}$ quelconques, $p_j = p$	NP-difficile	[49]
$m = 2$	$delay_{ii'}$, p_j quelconques	NP-difficile	[49]
		programme dynamique DPX ($O(n^2UB)$)	[49]
		schéma d'approximation $FPTAS$ ($O(\frac{n^3}{\varepsilon})$)	[49]
Fixé	$delay_{ii'} = d$, $p_j = p$	Algorithme $\mathcal{A}_1$ $O(n)$	[49]
	$delay_{ii'}$ quelconques, $p_j = p$	Algorithme $\mathcal{A'}_1$	[47]
	$delay_{ii'}$, p_j quelconques	NP-difficile	[48]
		heuristiques : (O.C.F, H_TR)	[48]
		($Aff\_tâches$, H_List)	[48]

L'étude de ce problème nous a inspiré vers de nouvelles perspectives de recherche, qui se résument dans les points suivants :

- Améliorer les heuristiques proposées et appliquer d'autres méthodes, en particulier dans la recherche de la meilleure séquence de machines qui se réduit au problème du chemin hamiltonien.
- Il serait intéressant d'étudier le problème avec d'autres contraintes portant sur les caractéristiques des tâches telles que les dates d'arrivées (r_j).
- Considérer le problème pour un autre critère $\overline{C} = \sum_j c_j$ par exemple, à voir même le cas multi-objectif.
- Voir l'analogie du problème avec les problèmes d'ordonnancement à temps réel.
- Traiter le problème en tenant compte des contraintes supplémentaires concernant le système de manutention. En effet, en considérant des contraintes sur la capacité des transporteurs et leur disponibilité, le problème se rapproche des problèmes de ramassage et livraison *Pickup and Delivery Problems* (PDP) qui constituent une classe importante des problèmes de tournées de véhicules *Vehicule Routing Problem* (VRP).

D'autres perspectives, en ce qui concerne l'étude de la complexité, peuvent aussi être considérées et qui peuvent donner suite à la généralisation des résultats obtenus pour les cas particuliers.

Bibliographie

[1] A. Agnetis, M. Flamini, G. Nicosia, A. Pacifici, Scheduling three chains on two parallel machines, European Journal of Operational Research, Vol. 202, Issue 3 : 669-674, 2010.

[2] A. Allahverdi, C. T. Ng, T. C. E. Cheng, M. Y. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal of Operational Research, Vol.187, Issue 3 : 985-1032, 2008.

[3] F. Afrati, E. Bampis, L. Finta, I. Milis, Scheduling trees with large communication delays on two identical processors, Journal of Scheduling, Vol. 8, N 2 : 179-190, 2005.

[4] KR. Baker, Introduction to sequencing and scheduling, John Wiley & Sons, New York, 1974.

[5] P. Baptiste, P. Brucker, Scheduling equal processing time jobs, In JY Leung (Ed.), Handbook of scheduling : algorithms, models and performance analysis, CRC Press, Boca Raton, FL, USA, 2004.

[6] P. Baptiste, Scheduling equal-length jobs on identical parallel machines, Discret Applied Mathematics, Vol. 103, Issue 1-3 : 21-32, 2000.

[7] S. Baruah, N. Cohen, C. Plaxton, D. Varvel, Proportionate progress : A notion of fairness in resource allocation, Algorithmica, Vol. 15, N 6 : 600- 625, 1996.

[8] J. Behnamian, M. Zandieh, S. M. T. Fatemi Ghomi, Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm, Expert System with Application, Vol. 36, Issue 6 : 9637-9644, 2009.

[9] J. E. Bell, P. R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Advanced Engineering Informatics, Vol. 18, Issue 1 : 41-48, 2004.

[10] C. Berge, Graphes, Gauthier-villars, 1983.

[11] J. Berit, Scheduling parallel jobs to minimize the makespan, Journal of Scheduling, Vol. 9 : 433-452, 2006.

[12] B. Bettayeb, I. Kacem, K. H. Adjallah, Ordonnancement sur machines parallèles identiques avec temps de préparation par famille : application à la gestion des tâches de maintenance préventive, Proceeding de la $6^{ème}$ Conférence Francophone de Modélisation et Simulation MOSIM'06, Rabat, Maroc, 3-5 Avril 2006.

[13] J. Blazewiez, M. Drozdowski, P. Formanowiez, W. Kubiak, G. Schmidt, Scheduling preemptable tasks on parallel processors with limited availability, Parallel Computing, 26 : 1195-1211, 2000.

[14] J. Blazewicz, Selected topics in scheduling theory, Annals of discrete mathematics, 31 : 1-60, 1987 .

[15] J. Blazewicz, KH. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Scheduling computer and manufacturing processes, Springer-Verlag, Berlin, 1996.

[16] J. Blazewicz, J. Lazewicz, KH. Ecker, G. Schmidt and J. Weglarz, Scheduling in computer and manufacturing systems, Springer-Lehrbuch, Berlin, 1994.

[17] C. Blum, Ant colony optimization : Introduction and recent trends, Physics of Life Reviws, Vol. 2, Issue 4 : 353-373, 2005.

[18] M. Boudhar, Sur quelques problèmes d'ordonnancement d'ateliers, thèse de magister en mathématiques spécialité recherche opérationnelle, U.S.T.H.B, 1991.

[19] M. Boudhar, Ordonnancement sur machines à traitement par batch sous contraintes de compatibilité de tâches : complexité et approches algorithmiques, thèse de doctorat d'état en mathématiques spécialité recherche opérationnelle, U.S.T.H.B, 2004.

[20] M. Boudhar, A. Haned, Ordonnancement préemptif sur machines parallèles identiques avec temps de changement de machines, Les annales ROAD : Recherche Opérationnelle et Aide à la Décision, N 28, 2008.

[21] M. Boudhar, A. Haned, Parallel machines scheduling with preemption and transportation time, Proceeding of the International Symposium on Operational Research, ISOR'08, 169-179, Alger, Algérie, 2-6 Novembre 2008.

[22] M. Boudhar, A. Haned, Preemptive scheduling in the presence of transportation times, Computers & Operations Research Vol. 36, N 8 : 2387-2393, 2009.

[23] P. Brucker, Scheduling algorithms, 5^{th} edition Springer Berlin Heidelberg, 2007.

[24] P. Brucker, S. Knust, Complexity results of scheduling problems, page web http ://www.mathematik.uni-osnabrueck.de/research/OR/class/, 2009.

[25] C. Çelik, I. Sariçiçek, Tabu search for parallel machines scheduling with job splitting, Sixth International Conference on Ininformation Technology, New Generation, 2009.

[26] I. Charon, A. Germa et O. Hudry : Méthodes d'optimisation combinatoire, édition Masson, 1995.

[27] T. C. E. Cheng, C. C. S. Sin, An algorithm for the $N|M|parallel|C_{max}$ preemptive due-date scheduling problem, Engineering Costs and Production Economics, Vol. 21, Issue 1 : 43-49, 1991.

[28] H. Cho, B. Ravindran, E. D. Jensen, An Optimal Real-Time Scheduling Algorithm for Multiprocessors, Proceedings of the 27^{th} IEEE International Real-Time Systems Symposium (RTSS'06), 101-110, Rio de Janiero, Brazil, 5-8 December 2006.

[29] C. Chu, J. M. Proth, L'ordonnancement et ses applications, edition Masson, 1996.

[30] R.I. Davis, A. Burns, A Survey of Hard Real-Time Scheduling for Multiprocessor Systems, ACM Computing Surveys, ACM Computing Surveys, DOI= 10.1145/1978802.1978814, Volume 43 Issue 4, October 2011.

[31] M. Dell'Amico, M. Iori, S. Martello : Heuristic algorithms and scatter search for the cardinality constrained $P||C_{max}$ problem, journal of Heuristics, 2004, Vol. 10, N 2 : 169-204, 2004.

[32] M. Demange, V. Paschos, Autour de nouvelles notions pour l'analyse des algorithmes d'approximation : formalisme unifié et classes d'approximation. RAIRO-Operations Research-Recherche Opérationnelle, 36, 3 : 237-277, 2002.

[33] M. L. Dertouzos, A. Ka-Lau Mok, Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks, IEEE Transaction on software engineering, Vol. 15, N 12 : 1497-1506, 1989.

[34] K. Djellab, Scheduling preemptive jobs with precedence constraints on parallel machines, European Journal of Operatioal Research, Vol. 117, Issue 2 : 355-367, 1999.

[35] M. Dorigo, L. M. Gambardella, Ant colonies for the traveling salesman problem, BioSystems, Vol. 43, Issue 2 : 73-81, 1997.

[36] M. Dorigo, G. Dicaro, L. M. Gambardella, Ant algorithm for discret optimization, Artificial life, 5 : 137-172, 1999.

[37] M. Dorigo, C. Blum, Ant colony optimization theory : A survey, Theoretical Computer Science, 344 : 243-278, 2005.

[38] M. Dorigo, T. St*ü*tzele, Ant Colony Optimization, A Bradford Book, 2004.

[39] S. Dunstall, A. Wirth, Heuristic method for the identical parallel machine flowtime problem with set-up times, Computer & Operations Research, 32 : 2479-2491, 2005.

[40] D. W. Engels, J. Feldman, D. R. Karger, M. Ruhl, Parallel processor scheduling with delay constraints, Proceeding of the 12^{th} annual ACM-SIAM symposium on Discrete algorithms SODA'01, Philadelphia, 577-585, 2001.

[41] B. Escoffier, Approximation polynomiale des problèmes d'optimisation : aspects structurels et opérationnels, thèse doctorat en informatique, France, 2005.

[42] A. Fishkin, K. Jansen, S. Sevastyanov, R. Sitters, Preemptive scheduling of independent jobs on identical parallel machines subject to migration delays, Lecture Notes of Computer Science, N 3669 : 580-591, 2005.

[43] P. Galinier, M. Habib, J. K. Hao : Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, Revue d'Intelligence Artificielle, Vol. 13, N 2 : 283-324, 1999.

[44] MR. Garey, DS. Johnson, Computers and intractability : a guide to the theory of NP-Completeness, W.H. Freeman and Company : San Francisco, 1979.

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

[46] A. Haned, Ordonnancement préemptif sur machines parallèles identiques avec temps de changement de machines, mémoire de magister en mathématiques spécialité recherche opérationnelle, U.S.T.H.B, 2007.

[47] A. Haned, M. Boudhar, A. Soukhal, N. Huynh Tuong, Preemptive Scheduling with jobs transportation, Proceeding of the 12th International Conference on Project Management and Scheduling, PMS'10, 217-221, Tours, France, 26-28 Avril 2010.

[48] A. Haned, M. Boudhar, A. Soukhal, Resolution of the parallel machines scheduling problem with preemption and transportation delays, Proceeding de la $8^{ème}$ édition du Colloque sur L'Optimisation et les Systèmes d'Information, COSI'11, 409- 420, Guelma, Algérie, 24-27 Avril 2011.

[49] A. Haned, A. Soukhal, M. Boudhar, N. H. Tuong, Scheduling on parallel machines with preemption and transportation delays. Computers & Operations Research, N 39 : 374-381, 2012.

[50] C. Hanen, A. Munier, An approximation algorithm for scheduling dependent tasks on m processors with small communication delays, Discret Applied Mathematics, Vol. 108, Issue 3 : 239-254, 2001.

[51] P. Holman, J. H. Anderson, Adapting p-fair scheduling for symetric multiprocessors, Journal of Embedded Computing, Vol. 1, N 4 : 543-564, 2005.

[52] J.A. Hoogeveen, J.K. Lenstra, B. Veltman, Three, four, five, six, or the Complexity of Scheduling with Communication Delays, Operations Research Letters, Vol. 16, Issue 3 : 129-137, 1994.

[53] H. Hoogeveen, G. J. Woeginger, Avery difficult scheduling problrm with communication delay, Operations Research Letters, Vol. 29, Issue 5 : 241-245, 2001.

[54] K. L. Huang, C. J. Liao, Ant colony optimization combined with taboo search for the job-shop scheduling problem, Computer & Operations Research, Vol. 35, Issue 4 : 1030-1046, 2008.

[55] K. Jansen, M. Mastrolilli, R. Solis-Oba, Approximation algorithms for flexible job shop problems, International journal of foundations of computer science, Vol. 16, N 2 : 361-379, 2005.

[56] M. Ji, T. C. E. Cheng, An FPTAS for parallel-machine scheduling under a grade of service provision to minimize makespan, Information Processing Letters, Vol. 108, Issue 4 : 171-174, 2008.

[57] E. S. Kim, C. S. Sung, I. S. Lee, Scheduling of parallel machines to minimize total completion time subject to s-precedence constraints, Computers & Operations Research, Vol. 36, Issue 3 : 698- 710, 2009.

[58] C. P. Koulamas, Scheduling two parallel semiautomatic machines to minimize machine interference, Computers & Operations Research, Vol. 23, Issue 10 : 945-956, 1996.

[59] M. Y. Kovalyov, Improving the complexities of approximation algorithms for optimization problems, Operations Research Letters, Vol. 17, Issue 2 : 85-87, 1995.

[60] A. Kozlov, To the conjecture on the minimum number of migrations in the optimal schedule for the $Pm|pmtn(delay = d)|C_{max}$ problem, Proceeding of the 10^{th} workshop on Models and Algorithms for Planning and Scheduling Problems, MAPSP 2011, 248-249, Nymburk, Czech Republic, June 19-24, 2011.

[61] C. Lee, Z. Chen, Machine scheduling with transportation considerations, Journal of scheduling, Vol. 4 : 3-24, 2001.

[62] R. Mc Naughton , Scheduling with deadline and loss functions, Management Science, Vol. 6, 1-12, 1959.

[63] R. Mellouli, C. Sadfi, I. Kacem, C. Chu, Ordonnancemnt sur machines parallèles avec contraintes d'indisponibilité, Proceeding de la $6^{ème}$ Conférence Francophone de Modélisation et Simulation MOSIM'06, Rabat, Maroc, 3-5 Avril 2006.

[64] M. Messaoudi Ouchene, A. Derbala, Adaptation d'un algorithme de fourmis pour la résolution du problème difficile d'ordonnancement $P|prec|C_{max}$, Proceeding of the International Symposium on Operational Research, ISOR'08, 667-677, Alger, Algérie, 2-6 Novembre 2008.

[65] L. Min, W. Cheng, A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines, Artificial intelligence in engineering, Vol. 13, Issue 4 : 399-403, 1999.

[66] E. Mokotoff, An exact algorithm for the identical parallel machine scheduling problem, European Journal of Operational Research, Vol. 152, Issue 3 : 758- 769, 2004.

[67] S. Morin, C. Gagné, M. Gravel, Ant colony optimization with a spacialized pheromone trail for the car-sequencing problem, European Journal of Operational Research, Vol. 197, Issue 3, 1185-1191, 2009.

[68] A. Moukrim, Optimal scheduling on parallel machines for a new order class, Operations Research Letters, 24 : 91-95, 1999.

[69] A. Moukrim, A. Quilliot, Optimal preemptive scheduling on a fixed number of identical parallel machines, Operations Research Letters, Vol. 33, Issue 2 : 143-150, 2005.

[70] R. R Muntz, E. G. Coffman, Optimal Preemptive Scheduling on Two-Processor Systems, IEEE Transactions on Computers, Vol. C-18, N 11 : 1014-1020, 1969.

[71] D. Nait Tahar , F. Yalaoui, C. Chu, A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times, International Journal of Production Economics, Vol. 99, Issues 1-2, 63-73, 2006.

[72] E. Néron, F. Tercinet, F. Sourd, Search tree based approaches for paralle machine scheduling, Computers & Operations Research, Vol. 35, Issue 4 : 1127-1137, 2008.

[73] M. Pfund, J. W. Fowler, A. Gadkari, Y. Chen, Scheduling jobs on parallel machines with setup times and ready times, Computers & Industrial Engineering, Vol. 54, Issue 4 : 764-782, 2008.

[74] M. L. Pinedo, Scheduling : Theory, Algorithms, and Systems, 3^{rd} edition, Springer, 2008.

[75] V. J. Rayward-Smith, The complexity of preemptive scheduling given interprocessor communication delays, Information Processing Letters, 25 : 123-125, 1987.

[76] G. Riotteau, O. Scalonni, E. Néron, Ordonnancement sur des machines identiques avec splitting et temps de préparation dépendant de la séquence, Proceeding de la $3^{ère}$ Conférence Francophone de Modélisation et Simulation "Conception, Analyse et Gestion des Systèmes Industriels" MOSIM'01, Troyes, France, 25-27 Avril 2001.

[77] M. H. Rothkopf, Scheduling independant tasks on parallel processors, Management Science, Vol. 12 : 438-447, 1966.

[78] M. Sakarovitch, Optimisation Combinatoire : graphes et programmation linéaire, Hermann, Paris, 1984.

[79] M. Sakarovitch, Optimisation Combinatoire : programmation discrète, Hermann, Paris, 1984.

[80] S. Shim, Y. Kim, A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property, Computer & Operations Research, Vol. 35, Issue 3 : 863-875, 2008.

[81] S. J. Shyu, B. M. T. Lin, P. Y. Yin, Application of ant colony optimization for no-wait flow-shop scheduling problem to minimize the total completion time, Computer & Industrial Engineering, Vol. 47, Issue 2-3 : 181-193, 2004.

[82] C. Solnon, Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization, European Journal of Operational Research, Vol. 191, Issue 3 : 1043-1055, 2008.

[83] A. Soukhal, P. Martineau, Resolution of a scheduling problem in a flowshop robotic cell, European Journal of Operational Research, Vol. 161, Issue 1 : 62-72, 2005.

[84] E. G. Talbi, Metaheuristics from design to implementation, A John Wiley & Sons, 2009.

[85] G. J. Woeginger, A comment on parallel-machine scheduling under a grade of service provision to minimize makespan, Vol. 109, Issue 7 : 341-342, 2009.

[86] W. Xing, J. Zhang, Parallel machine scheduling with splitting jobs, Discrete Applied Mathematics, Vol. 103, Issue 1-3 : 259-269, 2000.

[87] D. Xu, K. Sun, H. Li, Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan, Computers & Operations Research, Vol. 35, Issue 4 : 1344- 1349, 2008.

[88] B. Yagmahan, M.M. Yenisey, Ant colony optimization for multi-objective flow-shop scheduling problem, Computer & Industrial Engineering, Vol. 54, Issue 3 : 411-420, 2008.