N° d'ordre : 02 / 2007 - M / MT

الجمهورية الجزائرية الديمقراطية الشعبية

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

وزارة التعليم العالي والبحث العلمي

جامعة هواري بومدين للعلوم و التكنولوجيا

MÉMOIRE

Présenté pour obtenir le diplôme de
MAGISTER

En **Mathématiques**
Spécialité : **Recherche Opérationnelle (Génie Mathématique)**

Par

Amina HANED

Thème :

ORDONNANCEMENT PREEMPTIF SUR MACHINES PARALLELES IDENTIQUES AVEC TEMPS DE CHANGEMENT DE MACHINES

Soutenu le 08/10/2007, devant le jury composé de :

M. BERRACHEDI Abdelhafid, Professeur, USTHB,	Président
M. BOUDHAR Mourad, Maître de Conférences, USTHB,	Directeur de thèse
M. BOUROUBI Sadek, Maître de Conférences, USTHB,	Examinateur
M. CHAABANE Djamal, Maître de Conférences, USTHB,	Examinateur
M. OULAMARA Ammar, Maître de Conférences, EMNancy,	Examinateur

Remerciements

Je remercie dieu de m'avoir donné la foi, la santé et le courage d'amener à terme ce travail.

Du fond de mon cœur je remercie mes chers parents qui m'ont toujours guidé, encouragé et qui ont fait de leur mieux pour que j'en arrive là aujourd'hui.

Toute ma gratitude et reconnaissance à monsieur Mourad BOUDHAR, directeur de thèse, maître de conférences à l'U.S.T.H.B., de m'avoir proposé ce sujet, pour son aide précieuse, ses conseils, sa présence et sa patience.

Je remercie tous les membres du jury :

Monsieur Abdelhafid BERRACHEDI, professeur à l'U.S.T.H.B., de m'avoir fait l'honneur de présider ce jury.
Monsieur Sadek BOUROUBI, maître de conférences à l'U.S.T.H.B.,
Monsieur Djamal CHAABANE, maître de conférences à l'U.S.T.H.B.,
Monsieur Ammar OULAMARA, maître de conférences à Ecole des Mines de Nancy (France) d'avoir accepté de participer à ce jury, d'évaluer et de juger ce travail.

Mes remerciements vont également à tous mes professeurs de l'U.S.T.H.B., qui ont contribué à ma formation durant toutes ces années.

Je remercie mes sœurs, mes tantes et oncles, toutes mes cousines et cousins ainsi que tous mes ami(e)s pour leurs encouragements et leurs soutiens.

Enfin, je remercie toute personne qui a participé de près ou de loin à la réalisation de ce travail.

À vous tous MILLE MERCIS.

Résumé

Dans ce mémoire, nous nous sommes intéressés au problème d'ordonnancement avec minimisation de la date de fin de traitement (makespan) d'un ensemble de n tâches indépendantes et morcelables T_1,T_2,...,T_n sur m machines parallèles identiques M_1,M_2,...,M_m. Chaque tâche T_i ($i = \overline{1,n}$) possède un temps de traitement p_i, et le transport d'une tâche T_i interrompue d'une machine M_j à une autre machine $M_{j'}$ nécessite $c_{jj'}$ unités de temps.

Une modélisation mathématique du problème posé sous forme d'un programme linéaire en variables réelles et bivalentes est proposée, on montre aussi que le problème est NP- difficile ce qui nous conduit à la recherche de méthodes approchées afin de le résoudre.

Quelques sous-problèmes polynomiaux sont traités. Ainsi des heuristiques pour la résolution du problème principal sont décrites et analysées.

Mots clés : Machines parallèles, préemption, temps de changement de machines, heuristiques, complexité.

Abstract

In this thesis, we consider the problem of scheduling n independent jobs on m identical parallel machines with preemption, to minimize total completion time (makespan). Each job T_i ($i = \overline{1,n}$) has a processing time p_i, and the transportation of an interrupted job from a machine M_j to another machine $M_{j'}$ requires a $c_{jj'}$ units of time.

We propose a linear program with real and binary variables, and we proof that the problem is NP-hard.

Some sub problems are analyzed and solved by a polynomial algorithm, finally heuristic methods and lower bounds are developed to solve the first problem.

Keywords: Parallel machines, preemption, transportation time, heuristics, complexity.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

La recherche opérationnelle est une discipline dont l'objet est d'aider les gestionnaires à prendre des décisions en utilisant des méthodes scientifiques adaptées. Ses origines remontent à la deuxième guerre mondiale, où elle a été appliquée pour répondre aux questions d'ordre militaire. Depuis son domaine d'application s'est élargit pour toucher, entre autres, des problèmes économiques et industriels.

La théorie de l'ordonnancement est une branche de la recherche opérationnelle, elle consiste à la recherche de modèles mathématiques et de méthodes de résolution efficaces pour les problèmes posés. Notons qu'un problème d'ordonnancement consiste à allouer des ressources au cours du temps pour réaliser un ensemble d'activités, il permet aussi de répondre essentiellement aux questions fondamentales :

- Sur quelle ressource doit-on exécuter une tâche ?
- Quand faudra-il l'exécuter ?

Vu leur intérêt en industrie, les problèmes d'ordonnancement font objet de nombreux travaux scientifiques et constituent un domaine de recherches continues.
A travers ce mémoire, nous allons traiter un problème d'ordonnancement sur des machines parallèles identiques pour minimiser la plus grande date de fin de traitement (makespan) sous la contrainte qui prend en compte les temps de transport, d'une machine à une autre, des tâches interrompues. Ce type de problème peut se rencontrer dans un atelier de production où une pièce est traitée sur deux machines différentes, ce qui nécessite un temps de transport non négligeable dans beaucoup de cas pratiques et qui n'est pas pris en charge dans les modèles actuels. Il peut être rencontré aussi dans le cas des ateliers multi-sites où la fabrication d'un

produit s'effectue sur plusieurs sites éloignés les uns des autres, les produits semi-finis sont transportés d'un site à un autre dans un laps de temps bien déterminé.

Le présent mémoire est organisé en quatre chapitres :
Le premier présente les éléments essentiels pour la formulation des problèmes d'ordonnancement. En premier lieu nous donnons un rappel des définitions et concepts de base de la théorie d'ordonnancement et de la complexité. La fin de ce chapitre est dédiée aux différentes méthodes de résolution utilisées et qui sont classées en deux catégories : les méthodes exactes et les méthodes approchées.

Dans le second chapitre, nous passons en revue les différents travaux ayant traité des problèmes d'ordonnancement sur machines parallèles sous diverses contraintes et qui visent la minimisation du makespan. La deuxième partie de ce chapitre est consacrée à la modélisation et l'étude de la complexité de notre problème.

Des bornes inférieures et supérieures ainsi que des heuristiques de résolution avec des exemples illustratifs sont proposés dans le troisième chapitre.

Pour tester la performance des heuristiques développées, des expérimentations numériques font l'objet du quatrième et dernier chapitre.

Une conclusion générale achève ce travail, donnant une récapitulation des résultats obtenus, ainsi que des perspectives pour de nouvelles recherches.

Chapitre 1

Généralités

1.1 Problèmes d'ordonnancement : Définitions et notations

L'ordonnancement est un champ d'investigation qui a suscité l'intérêt de beaucoup de chercheurs. De nombreux problèmes sont identifiés ainsi que des techniques de résolution sont développées pour les résoudre.
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 [21].
Les problèmes d'ordonnancement se rencontrent souvent dans le milieu industriel, on parle de l'ordonnancement des ateliers qui consiste à exploiter au mieux des moyens limités (des machines) pour réaliser un ensemble de travaux (tâches) en respectant certaines contraintes et en optimisant un ou plusieurs critères. On peut les rencontrer aussi en génie civil pour le suivi des projets où on cherche souvent à déterminer la durée totale, en administration pour la gestion du personnel et l'emploi du temps, ou en informatique pour allouer des processeurs à l'exécution des programmes.

On va s'intéresser particulièrement aux problèmes d'ordonnancement d'ateliers, dans lesquels, quatre notions fondamentales interviennent, ce sont les travaux (jobs, ou tâches), les

ressources, les contraintes et les objectifs. Chacune de ces notions est définie comme suit :

1.1.1 Les tâches

Une tâche est un ensemble d'opérations qui doivent être exécutées, et qui nécessitent un certain nombre d'unités de temps (la durée de la tâche) et d'unités de chaque ressource.
Généralement, une tâche T_i est caractérisée par :

- une date de début d'exécution notée t_i,
- une durée opératoire notée p_i (processing time),
- une date de fin d'exécution notée c_i (completion time),
- une date de disponibilité (d'arrivée) notée r_i (ready time ou release date) : elle signifie qu'une tâche T_i ne puisse commencer son traitement avant la date r_i,
- une date échue notée d_i (due date), telle que la tâche T_i doit être achevée avant cette date,
- un poids associé à la tâche T_i noté w_i.

On peut illustrer une tâche T_i par le schéma suivant :

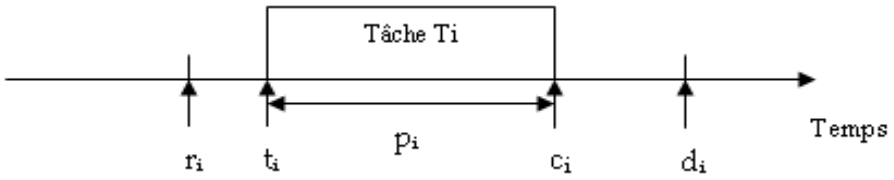

FIG. 1.1 – *Caractéristiques d'une tâche*

1.1.2 Les ressources

Une ressource est un moyen matériel ou humain mis en disposition pour la réalisation d'une tâche.
On distingue plusieurs types de ressources :

- ***Ressources renouvelables :*** elles restent disponibles en même quatité même après

avoir été utilisées par une ou plusieurs tâches, par exemple les hommes, les machines, l'équipement en général.

- ***Ressources consommables :*** telles que les matières premières dont la consommation globale est limitée au cours du temps.
- ***Ressources disjonctives (non partageables):*** exécutent qu'une seule tâche à la fois (robot manipulateur).
- ***Ressources cumulatives (partageables):*** peuvent être utilisées par plusieurs tâches simultanément (équipe d'ouvriers).

Dans notre cas on va considérer les machines. Une classification basée sur les différentes configurations des machines permet de distinguer entre les modèles suivants :

1. ***Machine unique :*** dans ce cas toutes les tâches sont traitées sur une seule machine. Ce cas peut être représenté par la figure ci-dessous.

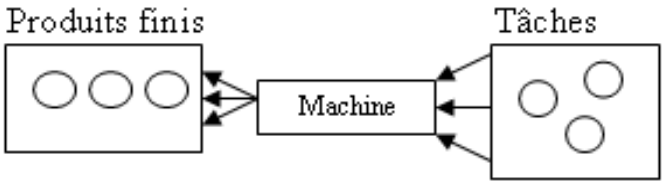

FIG. 1.2 – *Cas d'une seule machine*

2. ***Machines parallèles :*** exécutent les mêmes traitements et donc peuvent traiter n'importe quelle tâche. Et selon la vitesse de traitement on distingue :
   - Machines parallèles identiques qui ont toutes la même vitesse.
   - Machines parallèles uniformes, leurs vitesses sont différentes deux à deux mais restent indépendantes des tâches.
   - Machines parallèles non liées (différentes), pour lesquelles les vitesses sont différentes deux à deux et dépendent des tâches exécutées.

On peut schématiser le cas des machines parallèles par la figure suivante :

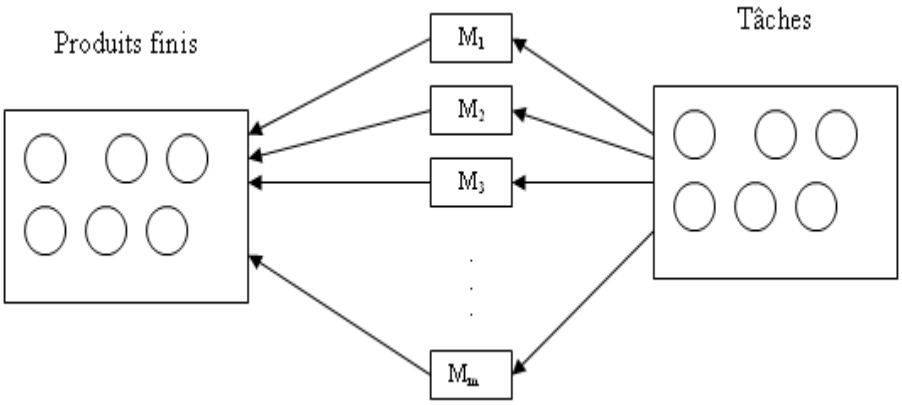

FIG. 1.3 – *Cas des machines parallèles*

3. ***Machines spécialisées:*** chaque machine est spécialisée à la réalisation de certaines tâches, dans ce cas les tâches sont composées d'opérations, chaque opération doit être exécutée par une machine spécifique, et selon le mode de passage des opérations sur les différentes machines, on différencie entre les trois systèmes ou modèles de traitement suivants :

– Flow-shop : Les tâches sont décomposées en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines selon un même ordre.

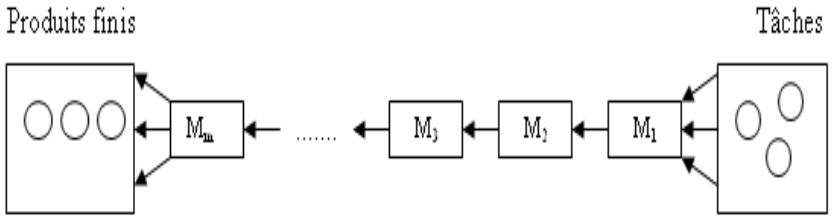

FIG. 1.4 – *Système flow-shop*

– Open-shop : les opérations des tâches sont exécutées sur toutes les machines dans n'importe quel ordre.

– Job-shop : les opérations des tâches doivent être exécutées sur un sous-ensemble de machines disposées en série, et peuvent avoir des routages différents.

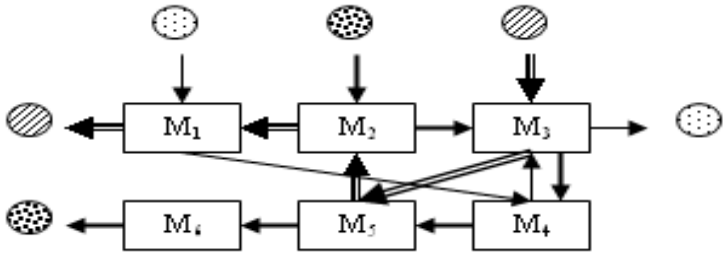

FIG. 1.5 – *Système job-shop*

En pratique d'autres modèles illustrant les problèmes industriels peuvent être définis. On peut citer à titre d'exemple les ateliers de type flow-shop hybride : ce sont des ateliers flow-shop dans lesquels un « étage » donné de la fabrication est assuré par plusieurs machines en parallèle. Dans ce genre d'ateliers, tout travail passe par chaque étage et l'ordre de passage sur les étages est le même [31].

1.1.3 Les contraintes

Les contraintes représentent les limites imposées par l'environnement ou les ressources, on les définit selon la nature du problème étudié. Elles intègrent essentiellement :

– les contraintes relatives aux dates limites des tâches ou à la durée totale d'un projet.

– les contraintes d'antériorité décrivant le positionnement de certaines tâches par rapport à d'autres.

D'autres contraintes portant sur la capacité des ressources, etc. peuvent être définies.

1.1.4 L'objectif

L'objectif est le critère d'optimisation ; c'est une évaluation numérique permettant d'apprécier 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 du système de la dernière tâche, $C_{max} = \max\limits_{1\leq i\leq n}\{c_i\}$, avec c_i date de fin d'exécution de la tâche T_i.
  La minimisation de la durée totale conduit à une répartition meilleure des charges en vue d'une utilisation rentable des machines [21].

- $\sum\limits_{i=1}^{n} w_i c_i$: 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, $L_{max} = \max\limits_{1\leq i\leq n}\{L_i\} = \max\limits_{1\leq i\leq n}\{c_i - d_i\}$,

- $\sum\limits_{i=1}^{n} w_i D_i$: somme pondérée des retards, $D_i = \max\limits_{1\leq i\leq n}\{c_i - d_i, 0\}$, qui permet d'évaluer les pénalités dues aux retards,

- $D_{max} = \max\limits_{1\leq i\leq n}\{D_i\} = \max\limits_{1\leq i\leq n}\{0, c_i - d_i\}$: le plus grand retard,

- $\sum\limits_{i=1}^{n} w_i U_i$: somme pondérée du nombre de tâches en retard.

Avec : $U_i = \begin{cases} 1 & \text{si la tâche } T_i \text{ est en retard;} \\ 0 & \text{sinon.} \end{cases}$

***Définition* 1.1.1.** *Un critère est dit régulier si pour deux ordonnancements différents S_1 et S_2, si S_1 est meilleur que S_2 alors au moins pour une tâche T_i on a : $c_{i1} < c_{i2}$ [21]*

***Définition* 1.1.2.** *Deux critères sont dits équivalents si une solution est optimale pour l'un est aussi optimale pour l'autre [21].*

La résolution d'un problème d'ordonnancement consiste alors à déterminer :

- le placement des tâches dans l'espace, c'est-à-dire sur les ressources,

– le placement des tâches dans le temps, c'est-à-dire l'instant de début d'exécution de chaque tâche sur les ressources qui participent à sa réalisation [26].

Dans de nombreux problèmes d'ordonnancement, 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 batch (lot) où une machine peut exécuter plusieurs tâches simultanément, et l'ordonnancement par chevauchements, où les opérations d'une même tâche peuvent êtres en cours d'exécution sur plusieurs machines en même temps. Dans la suite, on supposera que ces deux hypothèses sont vérifiées.

***Définition* 1.1.3.** *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 dira que le traitement est non préemptif.*

***Définition* 1.1.4.** *Un ordonnancement est dit sans attente si et seulement si la séquence des opérations composant toute tâche est exécutée sur les différentes machines sans aucune mise en attente. Il est dit sans temps mort (ou sans arrêt) si et seulement si aucune machine n'est mise en attente tant que toutes les tâches qui lui sont affectées ne sont pas encore traitées.*

1.2 Représentation d'un ordonnancement

Un ordonnancement est souvent représenté par un diagramme de Gantt. Celui-ci indique, selon une échelle temporelle, l'occupation des machines par les différentes tâches. A chaque tâche, on fait correspondre un segment horizontal de longueur proportionnelle à sa durée de traitement. On peut représenter aussi les différents temps morts, d'indisponibilité des machines, les temps de changement d'outils ou de machines, etc.

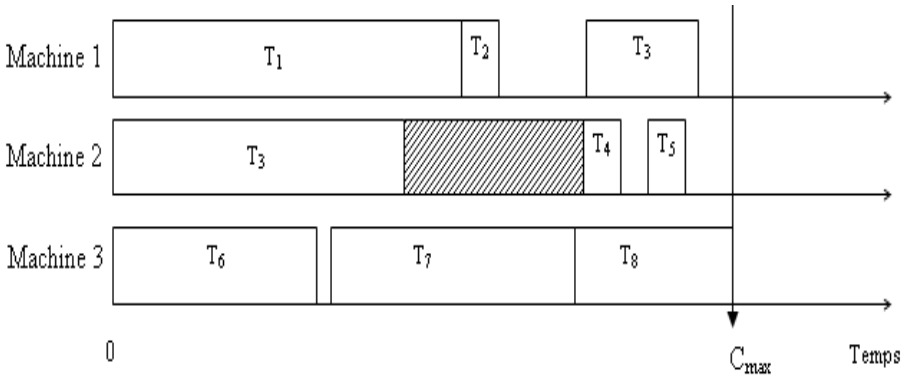

FIG. 1.6 – *Diagramme de Gantt*

1.3 Classification

Étant donné la diversité des problèmes d'ordonnancement, plusieurs classifications ont été proposées (voir [5], [26]). On va suivre celle proposée par Graham, elle comporte trois champs $\alpha/\beta/\gamma$.

1.3.1 Champ α

Associé à l'organisation des ressources, il est composé de deux (02) sous-champs α_1 α_2.

- α_1 représente le type de ressources (machines) utilisées, $\alpha_1 \in \{\emptyset,P,Q,R,F,O,J\}$.
  $\alpha_1 = \emptyset$: une seule machine, c'est le cas le plus simple qui peut être vu comme un cas particulier des autres,
  $\alpha_1 = P$: machines parallèles identiques,
  $\alpha_1 = Q$: machines parallèles uniformes,
  $\alpha_1 = R$: machines parallèles non liées (différentes),
  $\alpha_1 = F$: plusieurs machines spécialisées fonctionnant en mode flow-shop,
  $\alpha_1 = O$: plusieurs machines spécialisées fonctionnant en mode open-shop,
  $\alpha_1 = J$: plusieurs machines spécialisées fonctionnant en mode job-shop,

– α_2 dénote le nombre de machines composant le système, $\alpha_2 \in \{\emptyset, m\}$.
$\alpha_2 = \emptyset$ si le nombre de machines est variable,
$\alpha_2 = m$ si le nombre de machines est égal à m ($m > 0$).

1.3.2 Champ β

Décrit les différentes contraintes et caractéristiques des machines et des tâches. Il est composé de six (06) sous-champs β_1 β_2 β_3 β_4 β_5 β_6.

– β_1 indique le mode d'exécution, $\beta_1 \in \{\emptyset, pmtn\}$.
$\beta_1 = \emptyset$ indique que le mode est sans préemption,
$\beta_1 = pmtn$ indique que la préemption est autorisée.

– β_2 caractérise les ressources supplémentaires, $\beta_2 \in \{\emptyset, res\}$.
$\beta_2 = \emptyset$ indique qu'il n'y a pas de ressources complémentaires,
$\beta_2 = res$ spécifie qu'il y a des ressources complémentaires.

– β_3 décrit les contraintes de précédence entre les tâches, $\beta_3 \in \{\emptyset, prec, tree, chain\}$.
$\beta_3 = \emptyset$ reflète qu'il n'y a pas de contraintes de précédence entre les tâches,
$\beta_3 = prec$ indique qu'il y a des contraintes de précédence quelconques,
$\beta_3 = tree$ indique que les relations de précédence sont sous forme d'un arbre,
$\beta_3 = chain$ indique que les relations de précédence sont sous forme d'une chaîne.

– β_4 décrit les dates de disponibilité (d'arrivée) des tâches, $\beta_4 \in \{\emptyset, r_i\}$.
$\beta_4 = \emptyset$ lorsque toutes ces dates sont identiques ou égales à zéro,
$\beta_4 = r_i$ lorsqu'elles sont différentes.

– β_5 décrit les durées d'exécution des tâches, $\beta_5 \in \left\{\emptyset,\ p_i = p,\ \underline{p} \leq p_i \leq \overline{p}\right\}$

$\beta_5 = \emptyset$ indique que les tâches ont une durée arbitraire,

$\beta_5 = p_i = p$ lorsque toutes les tâches ont un temps d'exécution égale à p,

$\beta_5 = \underline{p} \leq p_i \leq \overline{p}$ lorsque les temps de traitement sont entre $\underline{p}$ et $\overline{p}$.

– β_6 indique les dates au plus tard des tâches pour lesquelles les tâches doivent être terminées, $\beta_6 \in \left\{\emptyset,\, d_i,\, \widetilde{d}_i\right\}$.

$\beta_6 = \emptyset$ si les tâches n'ont pas de date échue,

$\beta_6 = d_i$ lorsque chaque tâche T_i a une date échue d_i,

$\beta_6 = \widetilde{d}_i$ si chaque tâche T_i a une date limite qu'il faut absolument respecter.

D'autres sous-champs peuvent être ajoutés pour exprimer d'autres contraintes supplémentaires portant sur l'enchaînement des tâches, sur le mode de traitement, etc.

1.3.3 Champ γ

Approprié aux critères d'optimisation. Les objectifs visés sont liés à une bonne utilisation des ressources, une minimisation du délai global ou encore le respect d'un maximum de contraintes.

$$\gamma \in \left\{C_{max},\, L_{max},\, D_{max},\, \sum_{i=1}^{n} w_i C_i,\, \sum_{i=1}^{n} w_i D_i,\, \sum_{i=1}^{n} w_i U_i,\, ...\right\}$$

1.4 Efficacité des algorithmes et complexité des problèmes

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté des problèmes. Elle se concentre sur les problèmes qui peuvent être résolus effectivement, la question étant de savoir s'ils peuvent être résolus efficacement ou pas, en se basant sur une estimation (théorique) des temps de calcul et des besoins en mémoire informatique [31].

1.4.1 Classes P et NP

La théorie de la complexité repose sur la définition de classes de complexité qui permettent de classer les problèmes en fonction de la complexité des algorithmes qui existent pour les résoudre. Pour pouvoir exposer la notion de ces classes, il est nécessaire de donner quelques définitions de base.

***Définition* 1.4.1.** *Un algorithme est polynomial si le nombre d'opérations élémentaires nécessaires pour résoudre un exemple de taille n est borné par un polynôme en n. Un algorithme est efficace si et seulement si il est polynomial [28].*

***Définition* 1.4.2.** *Un problème de reconnaissance ou de décision est un problème dont la réponse est "vrai" ou "faux" [28].*

On peut associer à chaque problème d'optimisation un problème de décision, en définissant un seuil k pour la fonction objectif f, et on obtient le problème de décision " existe-il une solution réalisable S telle que $f(S) \leq k$?" pour un problème de minimisation, et $f(S) \geq k$? pour un problème de maximisation.

Définition* 1.4.3.** *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. De ce fait, les algorithmes non déterministes sont une construction abstraite et ne peuvent être mis en oeuvre sur ordinateur [28].*

***Définition* 1.4.4.** *La classe P regroupe tous les problèmes pouvant être résolus par un algorithme (déterministe) polynomial. Les problèmes de la classe P sont dits faciles [28].*

***Définition* 1.4.5.** *La classe NP est la classe qui regroupe les problèmes de décision résolus en temps polynomial par un algorithme non déterministe [28].*

Parmi les problèmes de la classe NP, les problèmes NP-complets, ils sont équivalents entre eux et s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de la classe NP.

Pour décrire cette classe d'équivalence, on définit la notion de réduction polynomiale entre deux problèmes.

***Définition* 1.4.6.** *Etant donnés deux problèmes de décision P1 et P2, on dit que P1 se réduit polynomialement à P2 s'il existe un algorithme pour P1 qui fait appel à un algorithme de résolution de P2, et qui est polynomial lorsque la résolution de P2 est comptabilisée comme une opération élémentaire [28].*

***Définition* 1.4.7.** *Un problème de décision est NP- complet si tout problème de la classe NP se réduit polynomialement à lui [28].*

***Définition* 1.4.8.** *Un problème d'optimisation dont le problème de décision associé est NP-complet est dit NP- difficile [28].*

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

1. montrer que D est dans la classe NP,
2. choisir D' un problème NP- complet connu,
3. construire une transformation f de D' à D,
4. montrer que f est une réduction polynomiale.

Cette preuve révèle une très grande importance puisque une fois la démonstration faite, on peut dire que l'existence d'un algorithme polynomial pour la résolution du problème considéré est peu probable, et par la suite les efforts seront dirigés pour la recherche d'une méthode de résolution approximative.

1.4.2 Problèmes NP-Complets connus

Parmi les problèmes NP-Complets célèbres, on cite :

- Problème SAT et sa variante 3-SAT.
- Problème du voyageur de commerce.
- Problème du cycle hamiltonien.

– Problème de la clique maximum.

– Problèmes de coloration de graphes.

– Problème de couverture des sommets dans un graphe.

Le problème P = NP est un problème ouvert posé en 1970. On a trivialement P $\subset$ NP car un algorithme déterministe est un algorithme non déterministe particulier. Mais la réciproque NP$\subset$ P, que l'on résume généralement à P=NP, est l'un des problèmes ouverts les plus fondamentaux et intéressants en mathématiques et informatique théorique. En effet, s'il s'avérait que P=NP, alors on pourrait résoudre tous les problèmes NP-complets en un temps polynomial.

1.4.3 Hiérarchie de complexité entre les problèmes d'ordonnancement

L'intérêt d'étudier les différentes relations entre les problèmes d'ordonnancement est d'appliquer des algorithmes de résolution de certains problèmes pour en résoudre d'autres qui leurs sont réductibles. Une représentation de ces relations peut avoir plusieurs configurations, selon le type de machines, les contraintes de précédence entre les tâches, les contraintes sur les durées de traitement ou bien selon les critères d'optimalité. Les figures ci-dessous représentent ces différentes configurations. Le sens de la flèche indique une réduction polynomiale.

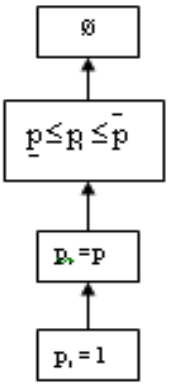

FIG. 1.7 – *Réduction entre les durées de traitement*

Les durées opératoires des tâches influent sur la complexité des problèmes, une simple contrainte sur les durées opératoires peut rendre le problème NP-difficile ou facile. Selon la figure 1.7, on voit que le cas où les temps de traitement sont tous égaux à 1 ($p_i = 1$) est un cas particulier de celui où ils sont constants ($p_i = p$) ou entre deux bornes ($\underline{P} \leq p_i \leq \overline{P}$) ou quelconques ($\varnothing$). Donc si le problème avec $p_i = 1$ est NP-difficile, alors le problème général avec $p_i = p$ l'est aussi. inversement, si le problème général avec p_i quelconques est polynomial alors les autres sous-problèmes le sont aussi.

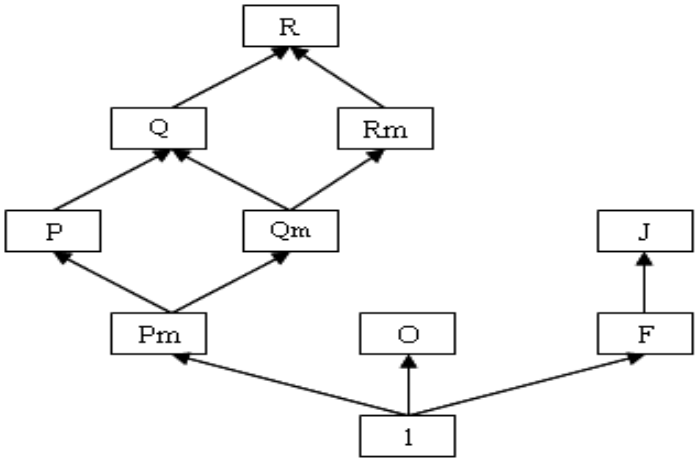

FIG. 1.8 – *Réduction entre les machines*

D'après le graphe, le cas le plus simple est celui où il n'y a qu'une seule machine, c'est un cas particulier de plusieurs machines parallèles identiques ou en série. La difficulté augmente si le nombre de machines est non borné et si on considère des machines uniforme ou non liées (différentes).

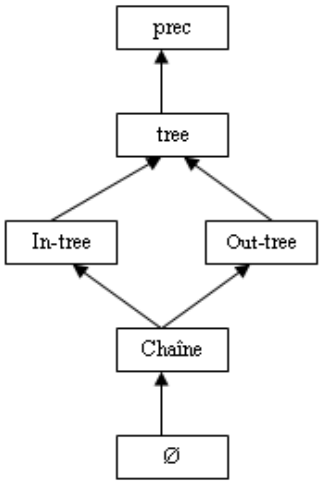

FIG. 1.9 – *Réduction entre les contraintes de précédence*

Selon la figure FIG. 1.9, il est nettement claire que le cas où il n'y a pas de contraintes de précédence est plus simple que celui où les contraintes sont considérées. La complexité varie selon leur nature (graphe de précédence des tâches).

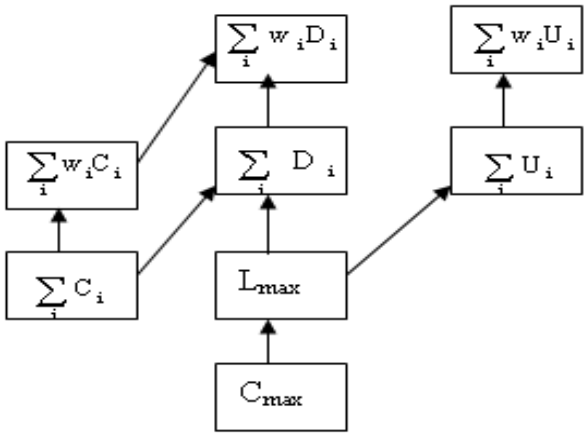

FIG. 1.10 – *Réduction entre les critères*

La dernière figure nous permet de voir la relation entre les principaux critères. D'autres représentations sont faites pour les différentes classes de problèmes, on peut se référer à [6].

1.5 Méthodes de résolution

Les problèmes d'ordonnancement font partie des problèmes d'optimisation combinatoire. Il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les cas.
Vu la grande importance des problèmes d'ordonnancement, de nombreuses méthodes de résolution ont été développées. La recherche opérationnelle est à l'origine de ces nombreuses méthodes qui peuvent être classées en deux grandes catégories : les méthodes exactes et les méthodes approchées [18].

1.5.1 Les méthodes exactes

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

1.5.1.1 Méthode par séparation et évaluation

Cette méthode consiste à développer une arborescence, dont la racine représente l'ensemble de toutes les solutions réalisables, en tenant compte d'un principe de séparation, un principe d'évaluation et une stratégie de développement. Le principe de séparation consiste à partager l'ensemble des solutions réalisables en sous ensembles en fonction d'un certain critère. L'évaluation permet de calculer (dans le cas d'un problème de minimisation) une borne inférieure de la solution obtenue après la séparation. Une borne supérieure est calculée au préalable, elle est utilisée pour éviter l'exploration des noeuds dont la valeur de la borne inférieure est supérieure à la valeur de la borne supérieure. La procédure de séparation est appliquée selon plusieurs stratégies, la première *en profondeur* consiste à descendre dans les branches jusqu'à ce qu'on trouve un sommet qu'on peut éliminer, ensuite on remonte pour descendre dans une autre branche. La seconde stratégie *le meilleur d'abord*, dans ce cas on commence par séparer les sommets qui ont l'évaluation la plus petite (pour un problème de

minimisation) en espérant trouver la solution optimale dans ces sommets. Enfin, on peut envisager une exploration en *largeur*, cette stratégie est peu utilisée pour des questions d'efficacité et d'encombrement de la mémoire [8].

1.5.1.2 La programmation dynamique

Introduite par Bellman dans les années 50, elle consiste à décomposer un problème de dimension n en n sous-problèmes de dimensions 1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se fait à partir des lois d'évolution du systéme et d'une décision. Le principe d'optimalité est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente [14].

1.5.2 Les méthodes approchées

Une méthode approchée ou heuristique est un algorithme qui a pour but de trouver une solution réalisable, tenant compte de la fonction économique, mais sans garantie d'optimalité, contrairement aux méthodes exactes qui trouvent toujours l'optimum si on leur laisse le temps qu'il faut. Il existe un très grand nombre d'heuristiques selon le problème à traiter. On distingue cependant les grands types suivants : méthodes constructives, recherches locales et les méta-heuristiques [20].

1.5.2.1 Méthodes constructives

Ce sont des méthodes itératives où à chaque itération, une solution est complétée. Ces méthodes sont généralement des algorithmes gloutons car elles considèrent les éléments dans un certain ordre sans remettre en question un choix une fois qu'il est fait. Pour les problèmes d'ordonnancement on peut citer les algorithmes de liste qui consistent à établir une liste, à partir d'un critère de priorité, puis construire l'ordonnancement à partir de cette liste.

1.5.2.2 Recherches locales

Ces méthodes sont initialisées par une solution réalisable obtenue par une méthode gloutonne ou générée aléatoirement, et recherchent à chaque itération 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. Le schéma général de ces méthodes est le suivant :
On considère un problème d'optimisation combinatoire (S, f), où S est l'ensemble des solutions réalisables et f la fonction objectif de S dans $\mathbb{R}$. Un algorithme de recherche locale a la structure générale suivante, dans laquelle $V(s)$ est un voisinage de la solution s :

$s :=$ une solution initiale de S ;

$z := f(s)$;

Tant qu'il existe s' dans $V(s)$ avec $f(s') < z$ (pour un problème de minimisation) faire

$s := s'$; $z := f(s')$;

Fin Tant que;

Sortir s et z.

1.5.2.3 Méta-heuristiques

Les méta-heuristiques constituent une classe de méthodes approchées adaptables à un très grand nombre de problèmes combinatoires. Elles ont révélé leur grande efficacité pour fournir des solutions approchées de bonne qualité [18]. Parmi ces méthodes, on trouve la méthode du recuit simulé, la recherche tabou, les algorithmes génétiques, etc. Dans ce qui va suivre, on donnera le principe de ces méthodes. Pour avoir plus de détails ainsi que des exemples d'application, on peut se référer à l'ouvrage d'I. Charon, A. Germa et O. Hudry [8] ou celui de P. Lacomme, C. Prins et M. Sevaux [20].

A. Recuit simulé

Le recuit simulé (simulated annealing) a été inventé par les physiciens Kirkpatrick, Gelatt et Vecchi en 1983. Le recuit s'inspire des méthodes de simulation de Metropolis développées aux années 50. L'analogie s'inspire du recuit des métaux en métallurgie : un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un minimum local pour un problème d'optimisation combinatoire. Si on le refroidit lentement, les atomes se réarrangent,

les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalente du minimum global pour un problème d'optimisation combinatoire.
Cette méthode part du même principe des méthodes de recherche locale, et elle a l'originalité d'accepter une solution voisine moins bonne que la solution courante avec une certaine probabilité pour échapper aux optimums locaux.

B. Méthodes tabou

Les méthodes tabou (tabu search) ont été inventées par Glover et Hansen en 1986. Leur principe est le suivant : à chaque itération le voisinage (complet ou partiel) de la solution courante est examiné et la solution minimisant l'augmentation du coût est sélectionnée. Pour éviter le phénomène de cyclage, la méthode interdit de revisiter une solution déjà visitée. Pour cela, une liste tabou contenant les solutions visitées est utilisée.

C. Algorithmes génétiques

Cette classe de méthodes a été inventée par Holland dans les années 70, pour imiter les phénomènes d'adaptation des êtres vivants. L'application aux problèmes d'optimisation a été développée ensuite par Goldberg en 1989. On part d'une population initiale de NS solutions aléatoires, chacune étant codée par une chaîne de caractères appelée *chromosome*. Chaque chromosome est muni d'une mesure d'adaptation qui peut être la fonction objectif.
Un algorithme génétique choisit des paires de chromosomes parents, en favorisant les plus adaptés, et engendre de nouvelles solutions (enfants) en appliquant des opérateurs de croisement et de mutation. On espère ainsi que les bonnes solutions vont échanger par croisement leurs caractéristiques et engendrer des solutions encore meilleures.

1.5.3 Evaluation des heuristiques

Les heuristiques n'offrent aucune garantie d'optimalité, elles peuvent trouver l'optimum pour certaines données, ou en être très éloignées. Supposons qu'on étudie un problème combinatoire pour lequel on dispose déjà d'une méthode exacte (optimale) de référence. Pour une heuristique H et une donnée d, on note $H(d)$ le coût de la solution heuristique et $OPT(p)$ le

coût optimal.
On appelle performance relative de H sur d le quotient : $R_H(\mathrm{d})=\frac{H(d)}{OPT(p)}$ pour un problème de minimisation, et $R_H(\mathrm{d})=\frac{OPT(p)}{H(d)}$ pour un problème de maximisation.
On peut aussi examiner la performance même si on ne dispose pas de la solution optimale pour une instance donnée d, en considérant $OPT(p)$ une borne inférieure pour un problème de minimisation et supérieure pour un problème de maximisation.

Face à une application concrète, le choix d'une méta-heuristique est difficile du moment que la comparaison s'avère complexe puisque la qualité de la solution trouvée dépend du temps d'exécution (nombre d'itérations) d'une part et des paramètres intervenant dans ces méthodes d'autre part. Cependant, quelque soit la méta-heuristique utilisée, elle permet de résoudre un problème d'optimisation combinatoire de taille importante.

Finalement, les méta-heuristiques présentent de nombreux avantages, on cite :

- Ce sont des méthodes générales applicables à une large classe de problèmes.
- Efficaces pour de nombreux problèmes.
- Fournissent une solution en un temps de calcul raisonnable.

Par contres, elles ont aussi un certain nombre d'inconvénients, elles ne garantissent pas l'optimalité de la solution.

L'analyse des problèmes d'ordonnancement se fait en plusieurs phases qu'on résume dans le schéma suivant :

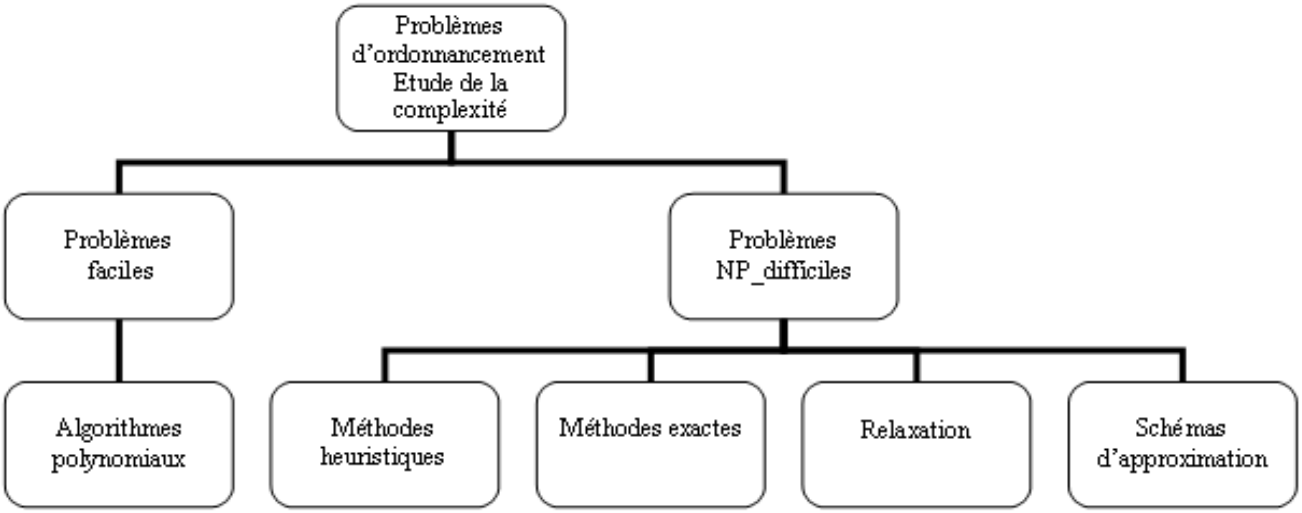

FIG. 1.11 – *Schéma récapitulatif de l'analyse des problèmes d'ordonnancement*

La relaxation consiste à relâcher certaines contraintes du problème qu'on cherche à résoudre. Quant aux schémas d'approximation c'est l'évaluation de la performance des méthodes de résolution proposées.

Chapitre 2

Position du problème et modélisation

2.1 Revues littéraires

Les problèmes à machines parallèles se caractérisent par le fait que plusieurs machines sont disponibles pour l'exécution d'un travail. D'un point de vu théorique ces problèmes sont une généralisation du problème à une machine [21]. Ce type de problèmes a suscité un grand intérêt en raison de ses applications dans les systèmes informatiques et les systèmes de production. Diverses contraintes et critères sont pris en considération ce qui a conduit à l'identification de nombreux problèmes. Nous allons citer quelques problèmes visant à minimiser la date de fin de traitement C_{max} (makespan).

Le premier travail concernant les problèmes d'ordonnancement sur machines parallèles a commencé à la fin des années 50 avec les travaux de McNaughton et Hu. Le problème P$//C_{max}$ est largement étudié, c'est un problème NP-difficile puisque le sous problème P2$//C_{max}$ l'est aussi. Parmi les heuristiques proposées pour le résoudre, les algorithmes de liste où les tâches sont affectées aux machines suivant l'ordre d'une liste construite à partir d'un certain critère de priorité, parmi lesquels la règle LPT (Longest Processing Time) qui consiste à ranger les tâches dans l'ordre décroissant de leur temps de traitement, et la règle SPT (Shortest Processing Time) qui range les tâches dans l'ordre croissant de leur temps de traitement. L. Min et C. Wu [23] proposent un algorithme génétique pour la résolution du problème. Quant à E. Mokotoff [24], il a proposé un algorithme exacte.

En relâchant la contrainte de non préemption, le problème devient facile. Il est noté P/pmtn/C_{max} et peut être résolu de manière polynomiale à l'aide de l'algorithme de Mc Naughton.
Des variantes du problème P//C_{max} sont également étudiées en posant des contraintes supplémentaires. Dell'Amico et Martello [13] ont traité le problème où chaque machine ne doit traiter qu'un certain nombre limité de tâches, ils l'ont noté P/ $nb < k/C_{max}$. Dans [3] le problème qui consiste à ordonnancer n tâches indépendantes, où chaque tâche T_i a un temps de traitement p_i, une date d'arrivée r_i et nécessite simultanément $m_i \leq m$ machines à chaque instant de son traitement est étudié dans le cas où la préemption est autorisée et non autorisée, les problèmes ainsi définis sont notés respectivement P/pmtn, r_i, m_i/C_{max} et P/ r_i, m_i/C_{max}.
En considérant les contraintes de précédence entre les tâches, le problème P/prec /C_{max} est NP-difficile, mais il existe des cas particuliers pour lesquels une résolution polynomiale est envisageable par exemple le problème P/ in-tree, p_i=1/C_{max} et P/ out-tree, p_i=1/C_{max}.

Considérons maintenant les problèmes avec des contraintes de temps de préparation appelé aussi temps de changement. Ils expriment le temps nécessaire pour passer de l'exécution d'une tâche à une autre. Ce temps représente le temps de réglage des machines, changement d'outils, de pièces, nettoyage de certaines parties des machines ou l'inspection des machines. Dans la littérature ces problèmes sont connus sous le nom de "problèmes d'ordonnancement avec des temps de changement".
C. Flipo- Dhamens [17] a passé en revue la littérature traitant ces problèmes avec des changements dépendant de la séquence des tâches traitées sur une même machine, et fait l'analogie entre ces problèmes pour la minimisation du C_{max} et le problème de tournées de véhicules.
F. Yalaoui et C. Chu [12] ont donné une heuristique pour la résolution du problème qui consiste à minimiser le makespan sur machines parallèles identiques avec temps de changement entre les tâches et possibilité de morcellement de celles-ci.
M. Boustta [5] a traité le cas d'une machine à injection multi-bras avec réglages multiples afin de minimiser le temps de fabrication, il a assimilé le problème au problème à machines parallèles identiques avec temps de réglage.
D'autres problèmes avec ce type de contraintes sont étudiés pour des critères différents à titre d'exemple on cite B. Bettayeb, I. Kacem, K. H. Adjallah [1] qui ont traité le problème avec des

temps de préparation par famille, c'est-à-dire que les tâches constituent des familles différentes et le passage d'une famille à une autre sur la même machine exige un temps de préparation indépendant de la séquences des tâches, dans le but de minimiser la somme pondérée des dates de fin de traitement. Ils ont proposé trois heuristiques pour la résolution et ils ont donné une application à la gestion des tâches de maintenance préventive, un problème similaire a été étudié dans [15].

Enfin on cite ci-dessous quelques problèmes pour les machines parallèles avec d'autres contraintes et pour divers critères :
M. Dupuy [16] a traité le problème à machines parallèles identiques avec des contraintes de flexibilité exprimant le fait qu'une tâche ne puisse être traiter que par une machine d'un sous ensemble de l'ensemble des machines pour la minimisation de la somme des tâches en retard, il a fait référence dans sa thèse à de nombreuses recherches réalisées sur les problèmes d'ordonnancement à machines parallèles.
M. Sevaux et P. Thomin [29] ont présenté une méthode de recherche tabou qui permet de résoudre le problème Pm/ r_i /$\sum_i w_i U_i$.
Le problème en présence d'une période d'indisponibilité pour chaque machine afin de minimiser la somme des dates de fin de traitement, noté Pm, h_{j1}/ / $\sum_i c_i$ a été étudié et résolu dans [10] .
Le problème P2/pmtn, prec, r_i, p_i=p/$\sum_i c_i$ a été traité dans [22].
Une méthode exacte et polynomiale a été présentée dans [2] pour le problème Pm/r_i, arborescence/$\sum_i w_i\ C_i$ ainsi que pour le problème Pm/$p_i = 1$, r_i, arborescence/ $\sum_i w_i\ C_i$ dans [19] .

D'après la littérature examinée, on constate que la plupart des problèmes prennent en compte les temps de préparation dépendant ou non de la séquence des tâches, et leur résolution se fait généralement par des heuristiques.
Le problème qu'on va étudier dans la section suivante tient compte des temps de changement de machines dans le cas où la préemption des tâches est autorisée.

2.2 Position du problème

Soit le problème qui consiste à ordonnancer un ensemble de n tâches $\{T_1,...,T_n\}$ indépendantes et préemptives de durée p_i $(i = \overline{1,\ n})$ sur m machines parallèles identiques $\{M_1,...,M_m\}$ afin de minimiser la durée totale (C_{max}). Si une tâche T_i traitée sur une machine M_j est interrompue pour terminer son traitement sur une autre machine $M_{j'}$ alors son transport nécessite $c_{jj'}$ unités de temps, qu'on va appeler temps de changement de machines.

On suppose d'une part 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 d'ordonnancement. D'autre part le temps de changement de machine $c_{jj'}$ ($\forall$ j,j' et $j \neq j'$) ne dépend pas des tâches.

Le problème ainsi défini est noté Pm /pmtn, $c_{jj'}$ /C_{max}.

Exemple 2.1

Considérons 10 tâches indépendantes, morcelables T_1,T_2,T_3,...,T_{10} à traiter sur 4 machines parallèles identiques, les durées de traitement des différentes tâches ainsi que les temps de changement de machines sont donnés dans les deux tableaux ci-dessous.

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	5	6	8	7	3	4	9	6	3	4

TAB. 2.1 – *Exemple 2.1 : durées de traitement*

$M_{j'}/M_j$	M_1	M_2	M_3	M_4
M_1	0	3	8	4
M_2	8	0	8	1
M_3	4	8	0	2
M_4	2	1	3	0

TAB. 2.2 – *Exemple 2.1 : temps de changement de machines*

Une solution réalisable du problème est représentée par :

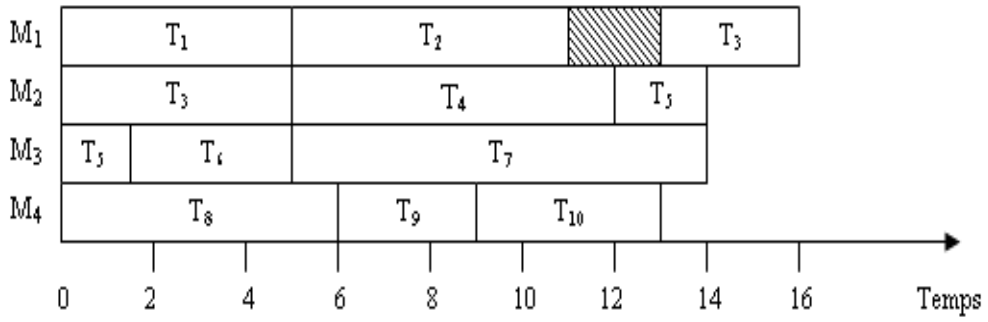

FIG. 2.1 – *Exemple 2.1 : solution*

Dans cet exemple, on a une solution réalisable de durée C_{max}=16. On remarque que les temps de changement de machines sont respectés, il y a des tâches qui n'ont pas été interrompues telles que T_1, T_2, T_4, T_6, T_7, T_8, T_9 et T_{10}. Les tâches interrompues sont T_3 et T_5.

L'industrie est un domaine où les problèmes d'ordonnancement apparaissent souvent, ils apportent un grand intérêt au bon fonctionnement des ateliers de production. Notre problème peut s'adapter à un atelier dans lequel le traitement d'une tâche (pièce) donnée s'effectue sur deux machines ce qui nécessite un temps non négligeable pour la transporter d'une machine à une autre (après avoir été interrompue). Les ateliers multi-sites sont aussi un bon exemple puisque la fabrication d'un produit se fait sur plusieurs sites différents et éloignés ce qui fait que le transport des produits semi-finis nécessite un certain temps.

On remarque que la matrice des temps de transport n'est pas symétrique dans le cas général puisque le transport d'une tâche donnée d'une machine M_j à une autre machine $M_{j'}$ n'est pas nécessairement le même. Le transport des tâches peut s'effectué 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.

2.3 Modélisation mathématique

On suppose que si une tâche est interrompue elle sera transportée sur une autre machine pour terminer son traitement, dans le cas contraire la préemption n'est plus intéressante on aura qu'à faire un décalage pour regrouper les différentes parts de la tâche intérrompue exécutées sur la même machine et les traiter l'une aprés l'autre (voir schéma ci-dessous).

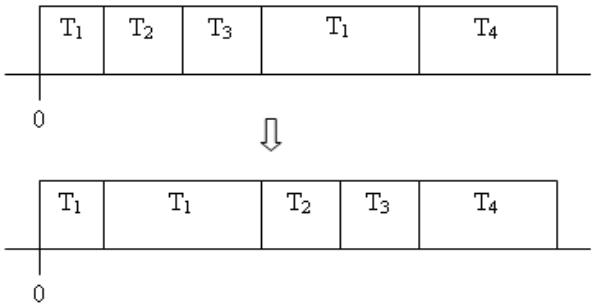

FIG. 2.2 – *Regroupement des opérations d'une même tâche*

2.3.1 Description des données et des variables

On définit :

n : le nombre de tâches ;

m : le nombre de machines ;

T : ensemble des tâches ; $T = \{T_1,...,T_n\}$;

M : ensemble des machines ; $M = \{M_1,...,M_m\}$;

p_i : temps de traitement de la tâche T_i, $p_i \in \mathbb{N}$ ($\forall\ i = \overline{1,n}$) ;

$c_{jj'}$: Le temps de transport nécessaire pour passer de la machine M_j à la machine $M_{j'}$, $c_{jj'} \in \mathbb{N}$;

y : période de la production ou durée de l'ordonnancement.

On définit les variables :

– t_{ij} : la date de début de traitement de la tâche T_i sur la machine M_j,
$t_{ij} \geq 0 \ \ \forall\ i = \overline{1,n},\ \ \forall\ j = \overline{1,m}$.

– Q_{ij} : la part du temps de traitement de la tâche T_i traitée sur la machine M_j, $Q_{ij} \geq 0 \;\; \forall\, i = \overline{1,n}, \;\; \forall\, j = \overline{1,m}$.

– $\alpha_{iji'j'}$ est définie comme suite:

$$\alpha_{iji'j'} = \begin{cases} 1 & \text{si } t_{ij} \leq t_{i'j'}; \\ 0 & \text{sinon.} \end{cases}$$

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

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

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

2.3.2 Description des contraintes

On doit tenir compte des contraintes suivantes :

– La somme de toutes les parts de traitement de chaque tâche T_i doit être égale au temps de traitement de la tâche, donc

$$\sum_{j=1}^{m} Q_{ij} = p_i, \quad \forall\, i = \overline{1,n};$$

– 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, donc

$$t_{ij} + Q_{ij} \leq y, \quad \forall\, i = \overline{1,n}, \;\; \forall\, j = \overline{1,m};$$

– 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 $t_{i'j} \leq t_{ij}$, on aura alors

$$\begin{cases} \mathrm{t}_{ij} + Q_{ij} - t_{i'j} \leq (1 - \alpha_{iji'j})C, \\ ou \qquad\qquad\qquad\qquad\qquad\qquad \forall\ i,\ i' = \overline{1,n} \text{ et } i \neq i',\ \forall\ j = \overline{1,m} \\ \mathrm{t}_{i'j} + Q_{i'j} - t_{ij} \leq \alpha_{iji'j}C, \end{cases}$$

Et on a aussi

$$\alpha_{iji'j} + \alpha_{i'jij} = 1, \qquad \forall\ i,i' = \overline{1,n} \text{ et } i \neq i',\quad \forall\ j = \overline{1,m}.$$

– Pour une tâche T_i interrompue sur la machine M_j et transportée vers la machine $M_{j'}$, soit la tâche 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 $t_{ij'} \leq t_{ij}$, on obtient alors

$$\begin{cases} \mathrm{t}_{ij} + Q_{ij} + x_{ij}c_{jj'} - t_{ij'} \leq (1 - \alpha_{ijij'})C, \\ ou \qquad\qquad\qquad\qquad\qquad\qquad \forall\ i = \overline{1,n},\ \forall\ j,\ j' = \overline{1,m} \text{ et } j \neq j' \\ \mathrm{t}_{ij'} + Q_{ij'} + x_{ij}c_{j'j} - t_{ij} \leq \alpha_{ijij'}C, \end{cases}$$

Et

$$\alpha_{ijij'} + \alpha_{ij'ij} = 1, \qquad \forall\ i = \overline{1,n},\quad \forall\ j,j' = \overline{1,m} \text{ et } j \neq j'.$$

– 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 obligatoirement 0, donc

$$x_{ij} \leq Q_{ij}C \qquad \forall i = \overline{1,n};\quad j = \overline{1,m}.$$

– 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. On aura donc

$$x_{ij}C \geq Q_{ij} \qquad \forall i = \overline{1,n};\quad j = \overline{1,m}.$$

2.3.3 Le critère à optimiser

On cherche à minimiser la date de fin de traitement makespan, donc $Miny$.

2.3.4 Le modèle

Enfin on obtient le modèle :

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

2.3.5 Taille du modèle

Notre modèle comporte :

- nm variables de type t_{ij},
- nm variables de type Q_{ij},

- nm variables de type x_{ij},
- $mn(n-1)$ variables de type $\alpha_{iji'j}$ pour j fixé et $i \neq i'$,
- $nm(m-1)$ variables de type $\alpha_{ijij'}$ pour i fixé et $j \neq j'$,
- une variable y.

Ce qui fait le total de $nm(n+m+1)+1$ variables.

Pour les contraintes on a :

- n contraintes de type (1),
- nm contraintes de type (2),
- $nm(n-1)$ contraintes de type (3),
- $nm(n-1)$ contraintes de type (4),
- $\frac{nm(n-1)}{2}$ contraintes de type (5),
- $nm(m-1)$ contraintes de type (6),
- $nm(m-1)$ contraintes de type (7),
- $\frac{nm(m-1)}{2}$ contraintes de type (8),
- nm contraintes de type (9),
- nm contraintes de type(10),

Donc $n + nm(\frac{5(n+m)}{2} - 2)$ contraintes.

Le modèle mathématique permet de trouver la solution exacte du problème posé pour des instances de petites tailles. Dans le cas des modèles linéaires avec des variables réelles, des solveurs tels que Lingo, CPLEX, etc. permettent de trouver cette solution. Le cas n'est pas aussi simple pour les modèles non linéaires et/ou avec des variables entières.

Rappelons que le modèle (P) possède $nm(n+m+1)+1$ variables et $n + nm(\frac{5(n+m)}{2} - 2)$ contraintes.
Par exemple, pour $n = 10$ et $m = 5$, on a 801 variables et 1785 contraintes.

2.4 Etude de la complexité

Nous allons montrer dans ce qui suit que le problème, avec seulement deux machines et des temps de changement de machines constants, est NP-difficile. Nous énonçons le théorème suivant :

***Théorème* 2.4.1.** *Le problème P2/ pmtn, $c_{jj'} = c/C_{max}$ est NP-difficile.*

Preuve

Nous utilisons une réduction du problème de 2-partition qui est NP-complet. Soit le problème de 2-partition suivant :

" Soient n entiers positifs a_1,...,a_n , existe-t-il un sous ensemble $J \subset I = \{1,...,n\}$, tel que $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$?"

Montrons que le problème de 2-partition se réduit polynomialement au problème d'ordonnancement O suivant :

" Étant données n tâches T_1,...,T_n, indépendantes et morcelables de durées $p_i = a_i$ ($\forall\ i = \overline{1,n}$), à traiter sur deux machines parallèles identiques M_1 et M_2. Le temps de changement de machines est $c_{jj'} = c = \frac{1}{2}\sum_{i=1}^{n} a_i$, $\forall\ j,\ j' \in \{1,\ 2\}$ et $j \neq j'$. On a aussi $c_{11} = \ c_{22} = 0$.
Soit $k = \frac{1}{2}\sum_{i=1}^{n} a_i$, "existe-t-il un ordonnancement des tâches T_1,...,T_n , sur M_1 et M_2, de durée $\leq k$?"

Le problème O est dans NP, en effet on peut vérifier en temps polynomial qu'une solution quelconque vérifie toutes les contraintes.

Supposons que le problème de 2-partition a une solution, nous avons donc un sous ensemble $J \subset I$, tel que : $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$.
A partir de cette solution, on peut construire un ordonnancement pour le problème O en traitant les $|J|$ tâches de J sur M_1 et les $|I \setminus J|$ tâches restantes sur M_2. Dans ce cas aucune tâche n'est interrompue, et par conséquent aucune tâche n'est transportée d'une machine à une autre,

donc la durée de l'ordonnancement est égale à $\frac{1}{2}\sum_{i=1}^{n} a_i$.

Supposons maintenant qu'il existe une solution pour le problème O de durée $\leq \frac{1}{2}\sum_{1}^{n} a_i$. Donc aucune tâche n'a été interrompue pour reprendre son traitement sur une autre machine car dans le cas contraire on aurait $C_{max} > \frac{1}{2}\sum_{i=1}^{n} a_i$ (on peut interrompre une tâche pour la reprendre sur la même machine, dans ce cas les temps de transport sont nuls et la préemption n'est pas intéressante).
Par conséquent chaque tâche n'est traitée que sur l'une des deux machines. Ainsi le problème de 2-partition a une solution en considérant l'ensemble J comme étant l'ensemble des tâches traitées sur M_1 et l'ensemble $I \setminus J$ celui des tâches traitées sur M_2. □

Par conséquent, le problème général Pm/ pmtn, $c_{jj'}$/ C_{max} est NP-difficile. Donc, il est très peu probable qu'il puisse exister un algorithme polynomial pour le résoudre.

Chapitre 3

Résolution

Les problèmes d'ordonnancement font partie des problèmes d'affectation en général, étant NP-difficiles, ils sont résolus par des méthodes approchées (heuristiques) ou par des méthodes exactes qui permettent de résoudre d'une manière optimale des problèmes de taille réduite.

Étant donné que le problème Pm /pmtn, $c_{jj'}$ /C_{max} est NP-difficile, la résolution par un algorithme polynomial est peu probable, pour cela on va tenter de le résoudre par des heuristiques.

Avant d'entamer la résolution, cherchons une borne inférieure et supérieure pour le makespan (C_{max}).

3.1 Détermination d'une borne inférieure

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}. Le premier terme $\max_{1\leq i\leq n}\{p_i\}$ reflète qu'une tâche donnée est traitée sur une seule machine à la fois. Le second terme $\frac{1}{m}\sum_{i=1}^{n} p_i$ représente la charge totale répartie sur les m machines, ou le temps de traitement moyen par machine.

***Proposition* 3.1.1.** *Si au moins une tâche est interrompue pour être traitée sur une autre machine, on propose la borne inférieure* $\overline{M'} = \min_{1\leq i\leq n}\{p_i\} + \min_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\}$

Preuve

Si une tâche T_i traitée sur la machine M_j est interrompue pour être traitée sur une autre machine $M_{j'}$ cela nécessite un temps de transport $c_{jj'}$. Donc $C_{max} \geq \min\limits_{1\leq i\leq n}\{p_i\} + \min\limits_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\}$.

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

Finalement $\overline{\overline{M}} = max\left\{\overline{M},\ \overline{M'}\right\}$ est une borne inférieure pour le C_{max}.

3.2 Détermination d'une borne supérieure

Comme borne supérieure, on peut penser à $\overline{S} = \sum\limits_{i=1}^{n} p_i$, si toutes les tâches sont traitées sur la même machine. Dans le cas contraire on a la proposition.

***Proposition* 3.2.1.** *Si au moins une tâche est interrompue pour être traitée sur une autre machine, alors* $\overline{S'} = \overline{M} + \max\limits_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\}$ *est une borne supérieure.*

Preuve

Considérons $\overline{M}$ la date de fin de traitement de l'ensemble des tâches si les temps de transport sont nuls. Si une tâche interrompue est transportée d'une machine M_j à une autre machine $M_{j'}$ avec une durée $c_{jj'}$, alors le temps de fin de traitement sera au plus égal à $\overline{M} + c_{jj'}$, d'où $\overline{M} + \max\limits_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\}$ est une borne supérieure.

3.3 Etude de quelques cas

Selon les différentes valeurs que peuvent prendre les temps de changement de machines à savoir les $c_{jj'}$, plusieurs cas se présentent.

3.3.1 1^{er} cas : les temps de transport sont nuls

Lorsque les temps de changement de machines sont nuls, c'est-à-dire $c_{jj'} = 0,\ \forall\ j,\ j'$, c'est le problème Pm / pmtn / C_{max}, qui est résolu par l'algorithme de Mc Naughton (1959) en un temps polynomial ($O(n)$).

Description de l'algorithme

L'algorithme permet de construire un ordonnancement en commençant par calculer la borne inférieure $\overline{M}$ sur la durée totale, telle que $\overline{M} = max\left\{\max_{1\leq i\leq n}\{p_i\}, \frac{1}{m}\sum_{i=1}^{n} p_i\right\}$, cette borne fournit une valeur optimale pour le C_{max}.

Une fois la borne déterminée, on sélectionne une tâche T_i, de l'ensemble des tâches non encore traitées, et la traiter sur la première machine à l'instant $t = 0$. On traite sur la même machine une autre tâche $T_{i'}$ non encore traitée à l'instant $(t+p_i)$, et ainsi de suite tant que $(t+p_i < \overline{M})$, sinon la tâche T_i est interrompue, on transfère le reste de cette tâche à la machine suivante.

Cet enchaînement est répété autant de fois jusqu'à ce que toutes les tâches soient traitées.

L'algorithme est le suivant :

Algorithme Mc Naughton [21];

Début

$\overline{M} := max\left\{\max_{1\leq i\leq n}\{p_i\}, \frac{1}{m}\sum_{i=1}^{n} p_i\right\}$;

$t := 0$; $i := 1$; $j := 1$;

Répéter

si$(t + p_i < \overline{M})$ **alors**

affecter la tâche T_i à la machine M_j à l'instant t ;

$t := t + p_i$; $i := i + 1$;

sinon si $(t + p_i = \overline{M})$ **alors**

affecter la tâche T_i à la machine M_j à l'instant t ;

$t := t + p_i$; $i := i + 1$; $j := j + 1$;

sinon traiter T_i sur la machine M_j à l'instant t pour $(\overline{M} - t)$ unités de temps et le reste de la tâche $(p_i - (\overline{M} - t))$ sur la machine suivante;

$p_i := p_i - (\overline{M} - t)$; $t := 0$; $j := j + 1$;

fsi;

fsi;

Jusqu'à i=n;

$C_{max} := \overline{M}$;

Fin.

Quand une tâche est morcelée, les deux morceaux (parts) de la tâche sont placés en début et fin de l' ordonnancement ce qui permet d'éviter les chevauchements. On remarque aussi que le nombre de préemptions est au plus égal à $(m-1)$ [7].

Exemple 3.3.1

On souhaite traiter 10 tâches sur 5 machines, les temps de traitement sont représentés par le tableau 3.1 ci-dessous.

Tâches	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	5	16	3	12	12	9	3	19	15	21

TAB. 3.1 – *Exemple 3.3.1 : durées de traitement*

En appliquant l'algorithme de Mc Naughton, on trouve la solution représentée par le diagramme de la figure ci-dessous, avec C_{max}=max {20,23}=23.

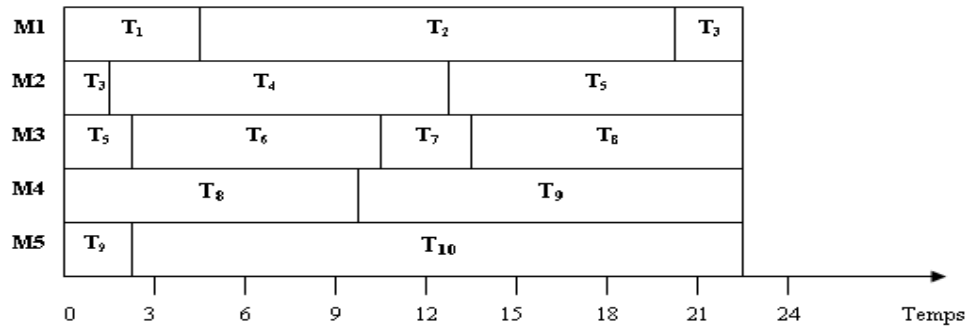

FIG. 3.1 – *Exemple 3.3.1 : solution générée par l'algorithme de Mc Naughton*

On remarque que les tâches T_3, T_5, T_8 et T_9 sont morcelées sur deux machines.

3.3.2 $2^{ème}$ cas : les temps de transport sont quelconques

Considérons la solution fournie par l'algorithme de Mc Naughton pour le problème Pm /pmtn /C_{max}, et soit $d_{ijj'} = \overline{M} - p_i$ l'écart entre la date de fin de traitement de la tâche T_i

interrompue sur la machine M_j et la date de son début de traitement sur la machine $M_{j'}$.

***Proposition* 3.3.1.** *Si $c_{jj'} \leq d_{ijj'}$, pour toute tâche interrompue T_i, alors la solution fournie par l'algorithme de Mc Naughton est optimale.*

Preuve

Soit $\overline{M}$ la valeur de la solution trouvée par l'algorithme de Mc Naughton, alors la date de début de traitement de toute tâche T_i $(\forall\ i = \overline{1,\ n})$ sur la machine $M_{j'}$, interrompue sur la machine M_j, ne sera pas affectée par le temps de transport de cette tâche vers la machine $M_{j'}$. Car après son transport la tâche reste en attente jusqu'à sa date de début de traitement sur la machine $M_{j'}$ comme le montre la figure FIG.3.2.

Comme $\overline{M}$ est une borne inférieure, et toutes les contraintes du problème sont vérifiées, en particulier les contraintes des temps de transport, alors $C_{max} = \overline{M}$ fournie une solution optimale.

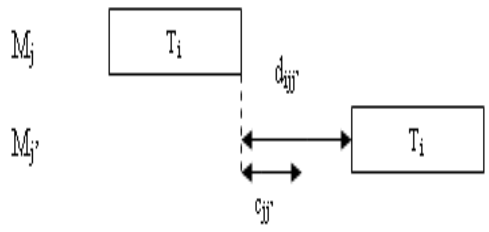

FIG. 3.2 – *Cas où $d_{ijj'} \geq c_{jj'}$ $\forall\ i,\ j,\ j'$*

3.3.3 $3^{ème}$ cas : les temps de transport sont constants

Nous supposons ici que $c_{jj'} = c\ \ \forall\ j,\ j' = \overline{1,\ m}$ et $j \neq j'$.

***Proposition* 3.3.2.** *Si $\overline{M} - p_i \geq c\ \ \forall i = \overline{1,\ n}$ avec $c_{jj'} = c\ \ \forall\ j,\ j' = \overline{1,\ m}$ et $j \neq j'$, alors l'algorithme de Mc Naughton donne une solution optimale.*

Preuve

Conséquence de la proposition 3.3.1.

Considérons maintemant le cas où $c_{jj'} = 1 \quad \forall\ j,\ j' = \overline{1,\ m}$ et $j \neq j'$, et soit l'algorithme suivant :

Algorithme Opt ;

Début

- Ranger les tâches selon la règle LPT ;
- Appliquer l'algorithme de Mc Naughton ;

Fin.

***Théorème* 3.3.1.** *L'algorithme Opt fournit une solution optimale pour le problème P/ pmtn,* $c_{jj'} = 1/\ C_{max}$ *en* $O(nlogn)$.

Preuve

En rangeant les tâches selon la règle LPT, les plus longues tâches sont traitées en premier. Donc si une tâche a une durée de traitement égale à $\overline{M}$, elle sera affectée entièrement (sans interruption) à une machine. Toutes les tâches interrompues ont une durée de traitement inférieures à $\overline{M}$. Ainsi $\overline{M} - p_i \geq 1$ pour toute tâche interrompue T_i. La solution obtenue est donc optimale. Une solution est générée en $O(nlogn)$ puisque le rangement des tâches nécessite l'utilisation d'un algorithme de tri (en général quicksort) qui s'exécute en $O(nlogn)$. □

3.3.4 Heuristiques de résolution

Lorsque les temps de transport $c_{jj'}$ sont quelconques, le problème est alors NP-difficile, on propose des heuristiques pour la résolution. Signalons que le cas où le nombres de machines est largement supérieur au nombre de tâches ($m \gg n$) est trivial, il suffit d'affecter chaque tâche à une machine libre, et dans ce cas $\overline{\overline{M}} = \max\limits_{1 \leq i \leq n} \{p_i\}$.

3.3.4.1 Description de l'heuristique H1

Une première heuristique inspirée de l'algorithme de Mc Naughton consiste à affecter les tâches à la première machine jusqu'à atteindre $\overline{\overline{M}} = max\left\{\overline{M},\overline{M'}\right\}$. Si la dernière tâche traitée sur la machine M_1 n'est pas achevée, elle sera affectée à la machine suivante à partir de $t = 0$ pour $p_i - (\overline{\overline{M}} - \overline{P_1})$ unités de temps. Si $p_i - (\overline{\overline{M}} - \overline{P_1}) + c_{21} > \overline{P_1}$ (où $\overline{P_1}$ est la date de début de la tâche T_i sur la machine M_1), alors on fait un décalage de telle sorte que la date de début de T_i sur M_1 soit égale à $p_i - (\overline{\overline{M}} - \overline{P_1}) + c_{21}$. On pose $\overline{\overline{M}} = p_i + c_{21}$. On poursuit le traitement des tâches non encore traitées sur la machine M_2, et si une tâche est interrompue on suit le même résonnement que précédemment.

L'heuristique compte essentiellement les étapes suivantes :

Algorithme H1;
Début
$\overline{\overline{M}} = max\left\{\overline{M},\ \overline{M'}\right\}$;
$t := 0\ ;\ i := 1\ ;\ j := 1\ ;\ k := 1\ ;$
Répéter
si $(t + p_i < \overline{\overline{M}})$ **alors**
affecter la tâche T_i à la machine M_j à l'instant t;
$d[j,k] := t\ ;\ c[j,k] := t + p_i\ ;$
//d[j,k] représente la date de début de la tâche T_i sur la machine M_j à la position k et c[j,k] sa date de fin de traitement
$t := t + p_i\ ;\ i := i + 1\ ;\ k := k + 1\ ;$
sinon si $(t + p_i = \overline{\overline{M}})$ **alors**
affecter la tâche T_i à la machine M_j à l'instant t;
$d[j,k] := t\ ;\ c[j,k] := t + p_i\ ;$
$h[j] := k\ ;$
//h[j] est le nombre total de tâches traitées sur la machine M_j
$i := i + 1\ ;\ j := j + 1\ ;\ t := 0;\ k := 1\ ;$

sinon traiter T_i sur la machine M_j à l'instant t pour $(\overline{\overline{M}} - t)$ unités de temps ;

$l := \overline{\overline{M}} - t$;

$d[j,k] := t$; $c[j,k] := \overline{\overline{M}}$;

$p_i := p_i - l$; $h[j] := k$;

$j := j + 1$; $t := 0$; $k := 1$;

traiter le reste de la tâche T_i sur la machine M_j à l'instant $t = 0$;

$d[j,k] := t$; $c[j,k] := t + p_i$;

$i := i + 1$; $k := k + 1$;

si $(c[j,k] + c_{jj-1} > d[j-1,h[j-1]])$ **alors**

changer la date de début sur la machine M_{j-1} ;

$d[j-1,h[j-1]] := c[j,k] + c_{jj-1}$;

$\overline{\overline{M}} := d[j-1,h[j-1]] + l$;

fsi;

fsi;

fsi;

Jusqu'à i=n;

$C_{max} := \overline{\overline{M}}$;

Fin.

Cette heuristique donne une solution réalisable en $O(n)$.

Exemple 3.3.2

On considère 10 tâches à traiter sur 5 machines, les durées opératoires des tâches ainsi que les temps de transport entre machines sont représentés dans les deux tableaux ci-dessous :

Tâches	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	5	16	3	12	12	9	3	19	15	21

TAB. 3.2 – *Exemple 3.3.2 : durées de traitement*

M_j/M_j	M_1	M_2	M_3	M_4	M_5
M_1	0	23	28	11	12
M_2	22	0	11	25	25
M_3	5	19	0	23	13
M_4	10	3	18	0	10
M_5	23	23	14	28	0

TAB. 3.3 – *Exemple 3.3.2 : temps de transport*

La borne inférieure $\overline{\overline{M}} = max\left\{max\left\{\max_{1\leq i\leq n}\{p_i\}, \frac{1}{m}\sum_{i=1}^{n} p_i\right\}, \min_{1\leq i\leq n}\{p_i\} + \min_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\}\right\}$

$\overline{\overline{M}} = max\{23{,}6\} = 23$.

Comme borne supérieure on a $\overline{S'} = max\left\{\max_{1\leq i\leq n}\{p_i\}, \frac{1}{m}\sum_{i=1}^{n} p_i\right\} + \max_{\substack{j,\ j'\\ j\neq j'}}\{c_{jj'}\} = 23{+}28{=}51.$

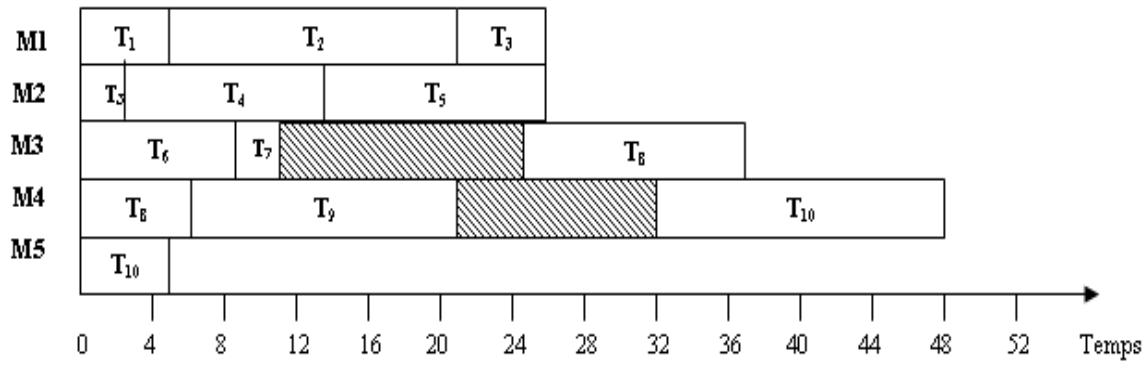

FIG. 3.3 – *Exemple 3.3.2 : solution générée par l'heuristique H1*

Remarquons que cette solution où $C_{max} = 48$ est réalisable du fait que toutes les contraintes sont respectées.

On remarque aussi que les décalages sont assez grands ce qui augmente les temps morts. Pour les réduire on peut envisager la non préemption des tâches dont le transport nécessite plus de temps que leur traitement en entier sur la même machine. En suivant ce principe on obtient la

nouvelle solution ci-dessous où $C_{max} = 36$.

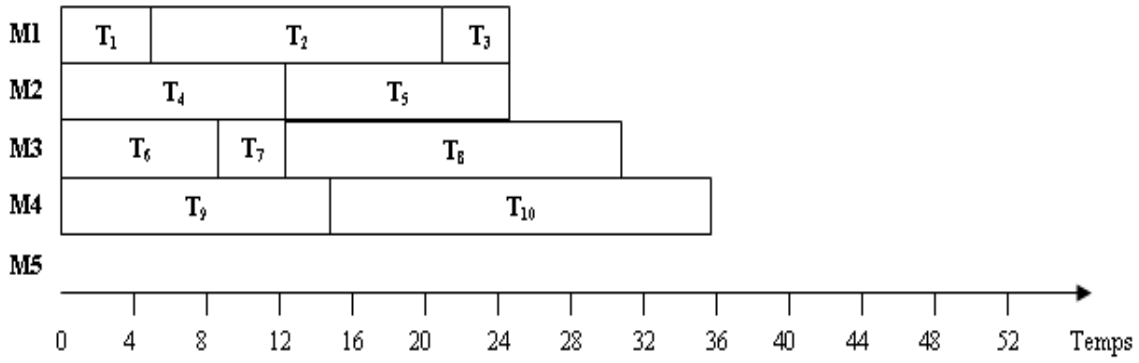

FIG. 3.4 – *Exemple 3.3.2 : solution générée par l'heuristique H1 modifiée*

Une autre version de cette heuristique qu'on appellera H1v1, peut être considérée, elle a le même principe que l'heuristique H1, la différence réside dans le fait que les machines sont rangées dans l'ordre croissant des temps de transport en commençant par la première ou par les deux machines nécessitant le minimum de temps de transport.

Afin de ranger les machines, on propose la procédure ci-dessous :

Procédure Minckl1

Début

1. déterminer le minimum de la première colonne de la matrice des temps de transport (puisque on commence par la première machine).

   Soit c_{k1} ce minimum. Le traitement des tâches va se poursuivre sur la machine M_k, et la tâche interrompue commence son traitement sur M_k et se termine sur M_1 ;
2. supprimer la première ligne et la première colonne ;
3. chercher le minimum de la colonne k, soit $c_{k'k}$ ce minimum ;
4. supprimer la ligne et la colonne k ;
5. poser $k := k'$ et aller à (3);
6. Répéter les étapes (3), (4) et (5) m-2 fois.

Fin.

Les indices des machines ainsi rangées seront sauvegardés dans un tableau (indice).
La complexité de cette procédure est $O(m^2)$, puisque à l'étape (1) on a m opérations, et pour chaque itération on a $(m-1)$,$(m-2)$,...,1 opérations respectivement ce qui fait $\frac{(m-1)m}{2}$ opérations. Donc la procédure est en $O(m^2)$.

3.3.4.2 Description de l'heuristique H1v1

Cette heuristique permet de construire un ordonnancement en commençant par ranger les machines par la procédure Minckl1, et puis traiter les tâches, sur la première machine dans l'ordre et en remplissant successivement et au maximum les machines jusqu'à atteindre la valeur de la borne inférieure. Si une tâche est interrompue on vérifie si le temps de changement est respecté sinon on fait un décalage. L'algorithme est le suivant :

Algorithme H1v1;

Début

Ranger les machines en faisant appel à la procédure Minckl1 ;

$\overline{\overline{M}} = max\left\{\overline{M}, \overline{M'}\right\}$;

$t := 0$; $i := 1$; $j := 1$; $k := 1$;

Répéter

si $(t + p_i < \overline{\overline{M}})$ **alors**

affecter la tâche T_i à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$t := t + p_i$; $i := i + 1$; $k := k + 1$;

sinon si $(t + p_i = \overline{\overline{M}})$ **alors**

affecter la tâche T_i à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$h[indice[j]] := k$;

$i := i + 1$; $j := j + 1$; $t := 0$; $k := 1$;

sinon traiter T_i sur la machine $M_{indice[j]}$ à l'instant t pour $(\overline{\overline{M}} - t)$ unités de temps ;

$l := \overline{\overline{M}} - t$;

$d[indice[j],k] := t$; $c[indice[j],k] := \overline{\overline{M}}$;

$p_i := p_i - l$;

$h[indice[j]] := k$;

$j := j + 1$; $t := 0$; $k := 1$;

traiter le reste de la tâche T_i sur la machine $M_{indice[j]}$ à l'instant $t = 0$;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$i := i + 1$; $k := k + 1$;

si $(c[indice[j],k] + c_{indice[j]indice[j-1]} > d[indice[j-1],h[indice[j-1]]])$ **alors**

changer la date de début sur la machine $M_{indice[j-1]}$;

$d[indice[j-1],h[indice[j-1]]] := c[indice[j],k] + c_{indice[j]indice[j-1]}$;

$\overline{\overline{M}} := d[indice[j-1],h[indice[j-1]]] + l$;

fsi;

fsi;

fsi;

Jusqu'à i=n;

$C_{max} := \overline{\overline{M}}$;

Fin.

L'heuristique H1v1 génère une solution réalisable en $O(m^2 + n)$.

Exemple 3.3.3

Reprenons l'exemple précédent. La solution est représentée par la figure ci-dessous, et la valeur de C_{max} est égale à 43. L'ordre des machines est : M_1,M_3,M_2,M_4,M_5.

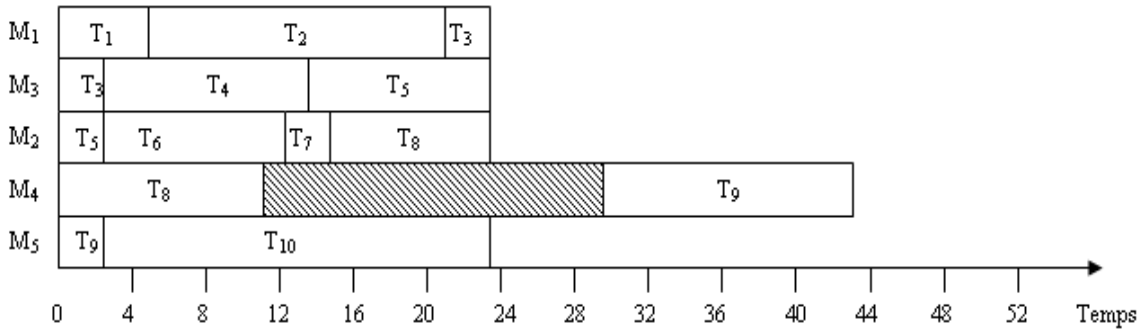

FIG. 3.5 – *Exemple 3.3.3 : solution générée par H1v1*

Essayons maintenant de commencer par les deux machines qui nécessitent le minimum de temps. Pour cela on propose la procédure Minckl2.

Procédure Minckl2

Début

1. déterminer le minimum de la matrice des temps de transport.
   Soit c_{kl} ce minimum ;
2. supprimer la ligne et la colonne l ;
3. chercher le minimum de la colonne k, soit $c_{k'k}$ ce minimum ;
4. supprimer la ligne et la colonne k ;
5. poser $k := k'$ et aller à (3) ;
6. Répéter les étapes (3), (4) et (5) jusqu'à élimination de toutes les machines.

Fin.

Les machines sont rangées en $O(m^2)$.

On définit la nouvelle heuristique H1v2, en faisant appelle à la procédure Minckl2 au lieu de Minckl1 dans l'heuristique H1v1.

3.3.4.3 Description de l'heuristique H1v2

Cette heuristique a le même principe que l'heuristique H1v1 sauf que les machines sont rangées par la procédure Minckl2.

Algorithme H1v2;

Début

- Ranger les machines en faisant appel à la procédure Minckl2 ;
- Appliquer l'heuristique H1v1 avec l'ordre trouvé par la procédure Minckl2;

Fin.

Cette heuristique génère une solution réalisable en $O(m^2 + n)$.

Exemple 3.3.4

En appliquant cette heuristique à notre exemple, on abouti à la nouvelle solution dont l'ordre des machines est : M_2, M_4, M_1, M_3, M_5. La valeur du C_{max} est égale à 29. La solution est représentée par la figure 3.6.

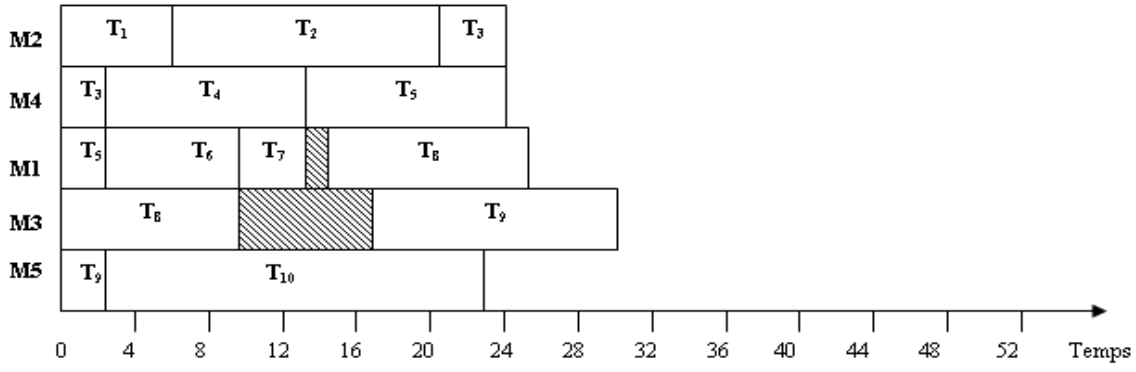

FIG. 3.6 – *Exemple 3.3.4 : solution générée par l'heuristique H1v2*

On remarque nettement que les temps morts sont réduits, les décalages sont moins importants et la solution générée est meilleure du fait que la valeur du C_{max} a diminué.

Dans les heuristiques précédentes les tâches sont prises dans leur ordre d'indice, on peut les ranger selon la règle LPT qui consiste à ranger les tâches dans l'ordre décroissant des temps de traitement puis appliquer l'une des heuristiques H1v1 ou H1v2.
Pour ranger les tâches, on utilise un algorithme de tri (généralement quicksort) qui permet de faire le tri en $O(nlogn)$.
les indices des tâches ainsi rangées seront sauvegardés dans un tableau (liste).

3.3.4.4 Description de l'heuristique H2

Cette nouvelle heuristique permet de déterminer un ordonnancement en affectant les tâches rangées selon la règle LPT aux différentes machines ordonnées par la procédure Minckl1 ou Minckl2. Les différentes étapes de cette heuristique sont données dans l'algorithme suivant :

Algorithme H2;

Début

Ranger les machines en faisant appel à la procédure Minckl2 (ou Minckl1) ;

Ranger les tâches selon la règle LPT ;

$\overline{\overline{M}} = max\left\{\overline{M}, \overline{M'}\right\}$;

$t := 0$; $i := 1$; $j := 1$; $k := 1$;

Répéter

si $(t + p_i < \overline{\overline{M}})$ **alors**

affecter la première tâche de la liste $T_{liste[i]}$ à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$t := t + p_i$; $i := i + 1$; $k := k + 1$;

sinon si $(t + p_i = \overline{\overline{M}})$ **alors**

affecter la tâche $T_{liste[i]}$ à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$h[indice[j]] := k$;

$i := i + 1$; $j := j + 1$; $t := 0$; $k := 1$;

sinon traiter $T_{liste[i]}$ sur la machine $M_{indice[j]}$ à l'instant t pour $(\overline{\overline{M}} - t)$ unités de temps ;

$l := \overline{\overline{M}} - t$;

$d[indice[j],k] := t$; $c[indice[j],k] := \overline{\overline{M}}$;

$p_i := p_i - l$;

$h[indice[j]] := k$;

$j := j + 1$; $t := 0$; $k := 1$;

traiter le reste de la tâche $T_{liste[i]}$ sur la machine $M_{indice[j]}$ à l'instant $t = 0$;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$i := i + 1$; $k := k + 1$;

si $(c[indice[j],k] + c_{indice[j]indice[j-1]} > d[indice[j-1],h[indice[j-1]]])$ **alors**

changer la date de début sur la machine $M_{indice[j-1]}$

$d[indice[j-1],h[indice[j-1]]] := c[indice[j],k] + c_{indice[j]indice[j-1]}$;

$\overline{\overline{M}} := d[indice[j-1],h[indice[j-1]]] + l$;

fsi;

fsi;

fsi;

Jusqu'à i=n;

$C_{max} := \overline{\overline{M}}$;

Fin.

L'heuristique H2 donne une solution en $O(nlogn + m^2)$.

En effet, le rangement des tâches se fait en $O(nlogn)$, l'ordre des machines en $O(m^2)$ et l'affectation des tâches en $O(n)$.

Exemple 3.3.5

On prend le même exemple traité précédement, la liste des tâches rangées est :

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

Pour ranger les machines on peut appliquer soit la procédure Minckl1 ou bien Minckl2. On obtiendra les résultats présentés par le tableau 3.4.

Heuristiques	H2v1 avec Minckl1	H2v2 avec Minckl2
Ordre des machines	M_1, M_3, M_2, M_4, M_5	M_2, M_4, M_1, M_3, M_5
Valeur du C_{max}	37	27

TAB. 3.4 – *Exemple 3.3.5 : solutions générées par l'heuristique H2*

Il est nettement clair qu'un rangement des tâches conduit à une solution meilleure, puisque

on commence par traiter les tâches de longues durées en premier lieu même si elles sont interrompues, leur transport ne prend pas beaucoup de temps du fait que les machines sont rangées dans l'ordre croissant des temps de transport. Les tâches de petites durées sont traitées vers la fin où elles peuvent être exécutées entièrement sans préemption, dans le cas contraire les écarts entre leur date de fin sur la machine où elles débutent et la machine où elles se terminent seront assez grands pour que les temps de transport seront respectés.

Afin d'éviter la préemption des tâches de longues durées et réduire les décalages, on propose l'heuristique H3.

3.3.4.5 Description de l'heuristique H3

Cette heuristique est légèrement différentes des autres, elle consiste à ranger les tâches selon la règle LPT dans un premier temps puis traiter les m premières tâches de la liste des tâches rangées sur les m machines à partir de $t = 0$. On traite les tâches restantes sur les machines rangées selon la procédure Minckl2, et si une tâche est interrompue on la place à la première position sur la machine M_j et à la dernière position sur $M_{j'}$ de cette façon l'écart sera assez grand pour que le temps de changement de machine soit respecté. L'algorithme est donné ci-dessous.

Algorithme H3;
Début
Ranger les machines en faisant appel à la procédure Minckl2 ;
Ranger les tâches selon la règle LPT ;
$\overline{\overline{M}} = max\left\{\overline{M}, \overline{M'}\right\}$;
$t := 0$; $k := 1$;
Pour $i := 1$ **à** m **faire**
affecter la tâche $T_{liste[i]}$ à la machine $M_{indice[i]}$ à l'instant t;
$d[indice[i],k] := t$; $c[indice[i],k] := t + p_{liste[i]}$;
$t := t + p_{liste[i]}$; $k := k + 1$;
fait;

$i := m + 1$; $j := 1$; $k := 2$; $t := c[indice[j],1]$;

Tant que $i \leq n$ **faire**

si $(t + p_{liste[i]} < \overline{\overline{M}})$**alors**

affecter la tâche $T_{liste[i]}$ à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_{liste[i]}$;

$t := t + p_{liste[i]}$; $i := i + 1$;

sinon si$(t + p_{liste[i]} = \overline{\overline{M}})$ **alors**

affecter la tâche $T_{liste[i]}$ à la machine $M_{indice[j]}$ à l'instant t ;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_{liste[i]}$;

$h[indice[j]] := k$;

$i := i + 1$; $j := j + 1$; $t := 0$; $k := 1$;

sinon traiter $T_{liste[i]}$ sur la machine $M_{indice[j]}$ à l'instant t pour $(\overline{\overline{M}} - t)$ unités de temps ;

$l := \overline{\overline{M}} - t$;

$d[indice[j],k] := t$; $c[indice[j],k] := \overline{\overline{M}}$;

$p_i := p_i - l$;

$h[indice[j]] := k$;

$j := j + 1$; $t := 0$; $k := 1$;

traiter le reste de la tâche $T_{liste[i]}$ sur la machine $M_{indice[j]}$ à l'instant $t = 0$, en permutant la tâche traitée à la première position avec la tâche $T_{liste[i]}$;

$d[indice[j],k] := t$; $c[indice[j],k] := t + p_i$;

$i := i + 1$; $k := k + 1$;

si $(c[indice[j],k] + c_{indice[j]indice[j-1]} > d[indice[j-1],h[indice[j-1]]])$ **alors**

changer la date de début sur la machine $M_{indice[j-1]}$;

$d[indice[j-1],h[indice[j-1]]] := c[indice[j],k] + c_{indice[j]indice[j-1]}$;

$\overline{\overline{M}} := d[indice[j-1],h[indice[j-1]]] + l$;

fsi;

fsi;

fsi;

fait;

$C_{max} := \overline{\overline{M}}$;

Fin.

Cette heuristique donne de bons résultats pour un nombre de tâches assez grand par rapport au nombre de machines, et une solution est générée en $O(nlogn + m^2)$

Exemple 3.3.6

Considerons un problème à 20 tâches et 5 machines.

T_i	T_1	T_2	T_4	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	3	5	6	3	5	7	4	2	4	5
T_i	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}
p_i	1	3	2	4	9	7	4	5	1	7

TAB. 3.5 – *Exemple 3.3.6 : durées de traitement*

M_j/M_j	M_1	M_2	M_3	M_4	M_5
M_1	0	15	5	3	16
M_2	15	0	14	7	29
M_3	22	5	0	2	30
M_4	1	18	14	0	6
M_5	9	17	17	19	0

TAB. 3.6 – *Exemple 3.3.6 : temps de transport*

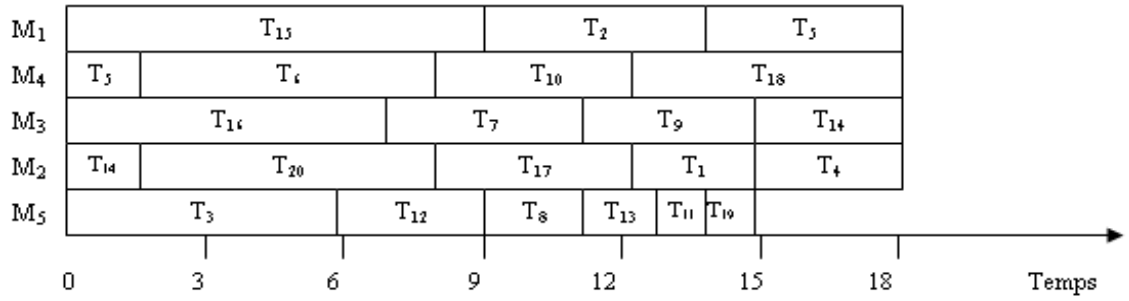

FIG. 3.7 – *Exemple 3.3.6 : solution générée par l'heuristique H3*

On remarque que cette solution est optimale puisque toutes les contraintes sont satisfaites et la valeur du C_{max} est égale à la valeur de la borne inférieure $\overline{\overline{M}} = 18$.

Remarque

Pour une solution trouvée en appliquant l'une des heuristiques précédentes, le nombre de préemptions par tâche est au plus égal à 1. Et au total (pour toutes les tâches) on a $(m-1)$ préemptions dans le plus mauvais cas.

En effet, pour une tâche donnée si elle est traitée à la première position, elle va être traitée entièrement sans interruption puisque la machine est libre, sinon elle sera interrompue et transportée sur une autre machine qui est libre puisque l'affectation des tâches se fait machine par machine, et donc la tâche interrompue aura suffisament de temps pour achever son traitement sans être interrompue une autre fois.

Chapitre 4

Expérimentations numériques

Afin d' évaluer la performance des heuristiques développées, plusieurs instances sont générées aléatoirement en utilisant des combinaisons différentes de tâches et de machines. Les tests sont effectués sur un Pentium (4) 1.80 GHz 256 Mo de RAM.

4.1 Génération des données

Les différents paramètres du problème (nombre de tâches et machines, temps de traitement et de transport) sont générés aléatoirement. Pour chaque instance le nombre de tâches n prend ses valeurs dans l'ensemble {10,20,50,100,200,300,500}, et le nombre de machines m est dans {5,10,15,20}.
Les temps de traitement des tâches ainsi que les temps de transport seront pris aléatoirement selon une loi uniforme, dans [1,30]et [1,100] respectivement.

4.2 Déroulement des tests

La première étape consiste à fixer le nombre de machines m, et pour chaque m fixé on fait varier le nombre de tâches n. On génère, plusieurs instances du même problème (100 instances pour chaque cas), on compte le nombre de fois où la solution trouvée par l'heuristique H. est meilleure par rapport aux autres, on compte aussi le nombre de fois où la valeur du C_{max} est

égale à la borne inférieure (dans ce cas la solution est optimale).

Les résultats obtenus sont représentés dans le tableau suivant, dans lequel on trouve le nombre de machines (colonne 1), le nombre de tâches (colonne 2), les autres colonnes donnent le nombre de fois où la solution trouvée par l'heuristique H. est meilleure par rapport aux autres solutions (sur la première ligne) ainsi que le nombre de fois où le C_{max} est égale à $\overline{\overline{M}}$ (sur la deuxième ligne).

Remarque

L'heuristique H1* est obtenue à partir de l'heuristique H1 en interdisant la préemption des tâches dont le transport nécessite beaucoup plus de temps que leur traitement en entier sur la même machine. En ce qui concerne l'heuristique H2v3 elle est obtenue en prenant les tâches dans l'ordre croissant de leur temps de traitement c'est-à-dire en les rangeant selon la règle SPT.

Rappelons aussi qu'on prend le nombre de tâche n supérieur au nombre de machines m, dans le cas contraire la solution du problème est triviale.

m	n	H1	H1*	H1v1	H2v1	H1v2	H2v2	H2v3	H3
5	10	3	57	9	12	25	11	2	/
		1	1	1	0	5	3	0	/
	20	12	38	24	27	28	23	16	41
		8	8	20	22	20	21	15	30
	30	31	37	54	60	68	69	47	80
		30	30	53	59	66	68	46	74
	50	100	100	100	100	100	100	86	100
		100	100	100	100	100	100	86	100

m	n	H1	H1*	H1v1	H2v1	H1v2	H2v2	H2v3	H3
10	20	0	8	15	9	47	29	9	/
		0	0	0	0	1	0	0	/
	30	0	9	12	5	30	11	3	49
		0	0	4	2	7	6	0	15
	50	1	3	39	42	60	55	21	70
		0	0	37	40	51	54	21	60
	100	100	100	100	100	100	100	73	100
		100	100	100	100	100	100	73	100
15	30	0	3	22	12	38	32	7	/
		0	0	0	0	1	0	0	/
	50	0	2	22	10	36	16	3	48
		0	0	11	8	15	11	1	24
	100	9	9	81	77	97	95	37	97
		9	9	81	77	97	95	37	97
	200	100	100	100	100	100	100	60	100
		100	100	100	100	100	100	60	100
20	50	0	0	13	11	44	44	2	/
		0	0	1	0	0	1	0	/
	100	0	1	39	45	68	64	11	75
		0	0	38	44	61	63	11	69
	200	100	100	100	100	100	100	58	100
		100	100	100	100	100	100	58	100
	300	100	100	100	100	100	100	50	100
		100	100	100	100	100	100	50	100
	500	100	100	100	100	100	100	51	100
		100	100	100	100	100	100	51	100
Total 1		656	767	930	910	1141	1049	535	1060
Total 2		648	648	846	852	924	922	509	969

TAB. 4.1 – *Résultats des tests*

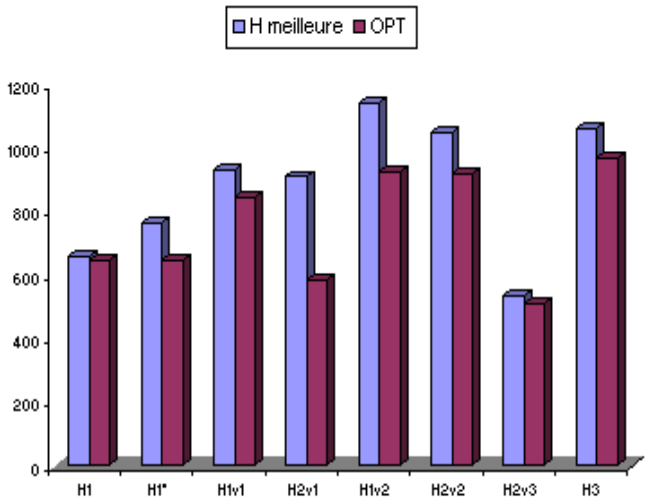

FIG. 4.1 – *Réultats des tests*

D'après les résultats présentés dans le tableau 4.1 et la figure 4.1, les trois heuristiques H1v2, H2v2 et H3 semblent meilleures. Pour $m = 5$ et $n = 10$; $m = 10$ et $n = 20$; $m = 15$ et $n = 30$ c'est l'heuristique H1v2 avec 25 % ; 47 %; 38 % respectivement, mais rien ne garantit l'optimalité des solutions. Pour les autres cas c'est l'heuristique H3 qui donne de bonnes solutions avec des pourcentages importants et on peut même voir que les solutions sont optimales pour 969 cas sur 1060 cas où elle est meilleure.

D'autres instances ont été générées, les résultats obtenus sont donnés par le tableau ci-dessous :

	H1	H1*	H1v1	H2v1	H1v2	H2v2	H2v3	H3
H. meilleure	2894	3030	3352	3373	4229	4318	2623	3787
Optimum	2880	2880	3218	3257	3885	3906	2566	3568

TAB. 4.2 – *Récapitulation de tous les tests*

4.3 Analyse des résultats

L'analyse consiste à comparer les heuristiques les unes par rapport aux autres ainsi qu'à la borne inférieure. Le tableau 4.2 résume les différents résultats qu'on peut schématiser par le graphique ci-dessous.

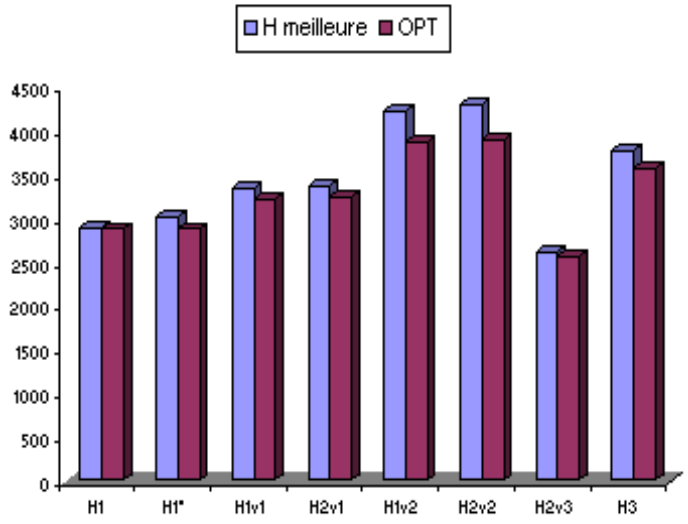

FIG. 4.2 – *Comparaison des heuristiques*

Les différents tests révèlent que dans la plupart des cas les heuristiques H1v2 et H2v2 fournissent de bonnes solutions pour les différentes combinaisons de machines et de tâches. En effet, l'heuristique H1v2 est meilleure dans 4229 cas, elle donne des solutions optimales dans 3885 cas. Quant à l'heuristique H2v2 les solutions générées sont optimales dans 3906 cas sur 4318 où elle est meilleure.

En ce qui concerne l'heuristique H3, elle est appliquée lorsque n le nombre de tâches est assez grand par rapport à m le nombre de machines. En comparant les solutions générées par H3 on constate qu'elle est meilleure que H2v2.

On constate aussi que pour les problèmes de petites tailles ($m = 5$ et $n = 10$ ou 20) les heuristiques H2v2 et H1v2 donnent de bonnes solutions. Lorsque n est assez grand, toutes les heuristiques donnent des solutions proches (la même valeur pour le C_{max}), il n'y a que la séquence des tâches et l'ordre des machines qui changent et dans la plupart des cas la solution

est optimale. Ceci peut être expliquer par le fait qu'un grand nombre de tâches augmente le nombre moyen de tâches par machine, et par suite, il fait augmenter l'écart entre la date de fin de traitement d'une partie d'une tâche interrompue traitée à la première position sur une machine M_j et sa date de début de traitement à la dernière position sur la machine $M_{j'}$. Si les temps de transport ne sont pas assez grands par rapport aux temps de traitement, ils seront respectés dans la plupart des cas sans faire de décalages.

Remarque

On remarque que la longueur de l'ordonnancement dépend des temps de changement de machines et des durées opératoires des tâches. Si les durées opératoires des tâches sont assez petites par rapport aux temps de transport il y aura des décalages importants et donc des temps morts considérables sur les machines. Dans le cas contraire et avec un grand nombre de tâches les temps de transport vont être respecter et donc les temps morts seront nuls.

Finalement, on peut dire que l'heuristique H2v2 basée sur le rangement des tâches selon la règle LPT avec des machines ordonnées conduit généralement à de bonnes solutions pour les problèmes de petites tailles. En ce qui concerne les problèmes de grandes tailles ils peuvent être résolus par l'heuristique H2v2. Cependant l'heuristique H3 est favorisée du fait qu'elle évite d'interrompre les tâches de longues durées.

4.4 Comportement des heuristiques par rapport à l'optimum

Dans cette phase, on compare les solutions générées par les heuristiques H2v2 et H3, qui sont jugées meilleures par rapport aux autres d'après les résultats des tests précédents, avec les solutions exactes trouvées à l'aide du logiciel Lingo.

Pour ce faire, nous avons généré aléatoirement quelques instances de taille réduite.
Le tableau ci-dessous résume les résultats obtenus :

Instances	H2v2	H3	Optimum	R_{H2v2}	R_{H3}
I1	24	21	21	1.142	1
I2	12	12	11	1.090	1.09
I3	17	17	17	1	1
I4	12	10	8	1.5	1.25
I5	23	22	22	1.045	1
I6	10	10	10	1	1
I7	22	20	20	1.1	1
I8	16	13	11	1.454	1.181
I9	10	10	10	1	1
I10	20	20	20	1	1

TAB. 4.3 – *Comportement des heuristiques par rapport à l'optimum*

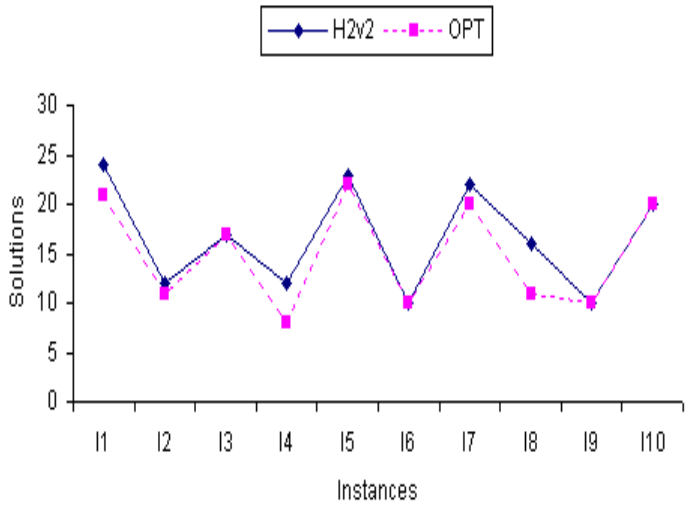

FIG. 4.3 – *Comparaison des solutions générées par heuristique H2v2 et l'optimum*

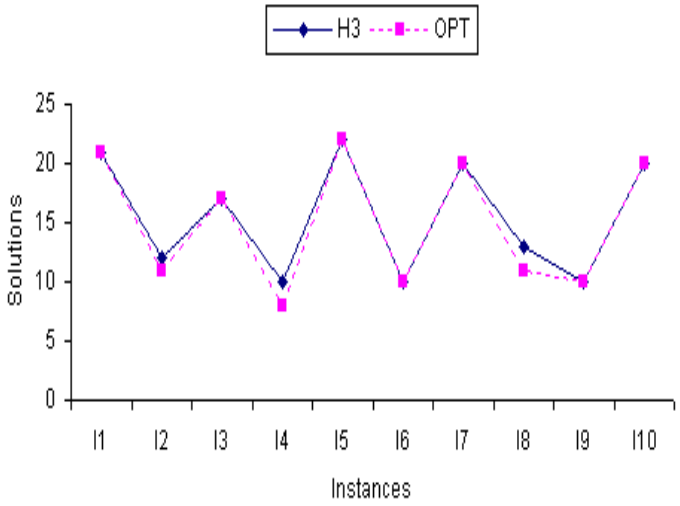

FIG. 4.4 – *Comparaison des solutions générées par heuristique H3 et l'optimum*

En comparant les deux courbes représentatives des solutions générées par les heuristiques H2v2 et H3, on voit clairement que les solutions générées par l'heuristique H3 sont assez proches de la solution optimale ce qui nous conduit à dire que l'heuristique H3 est meilleure.

Examinons maintemant les rapports de performance des deux heuristiques. On voit clairement que le rapport de performance de l'heuristique H3 est plus proche de 1, ne qui nous confirme que H3 est meilleure que H2v2.

4.5 Comportement des heuristiques par rapport à la borne inférieure

Dans ce cas, on génère des instances pour des problèmes de taille considérable dont on ne possède pas de solution optimale.
L'analyse porte sur la comparaison des solutions trouvées par les heuristiques avec la borne inférieure. Les résultats sont donnés ci-dessous :

Instances	H2v2	H3	$\overline{\overline{M}}$
I1	155	155	155
I2	156	156	156
I3	145	145	145
I4	73	73	73
I5	60	60	60
I6	144	144	144
I7	161	161	161
I8	157	157	157
I9	149	149	149
I10	164	164	164

TAB. 4.4 – *Comportement des heuristiques par rapport à la borne inférieure*

On remarque qu'à chaque fois la borne inférieure est atteinte, ceci est dû au fait d'avoir un très grand nombre de tâches fait augmenter le nombre moyen de tâches par machine comme on l'a expliqué précédemment.

En conclusion, les deux heuristiques donnent des solutions de bonne qualité, mais l'heuristique H3 est favorisée puisque elle évite la préemption des tâches de grandes durées opératoire.

Conclusion

Le travail présenté dans ce mémoire s'intéresse à l'étude d'un problème d'ordonnancement sur machines parallèles identiques en tenant compte des temps de transport des tâches interrompues, pour la minimisation du temps de fin de traitement (makespan).

Dans ce mémoire, nous avons présenté brièvement l'étude des problèmes d'ordonnancement, en premier lieu nous avons donné quelques éléments de base concernant la terminologie d'ordonnancement, la classification des différents problèmes, la complexité des algorithmes et nous avons exposé ensuite les différentes méthodes de résolution.

Un état de l'art des principaux travaux traitant les problèmes d'ordonnancement sur des machines parallèles avec de multiples contraintes est donné en introduction du deuxième chapitre. Le problème étudié est ensuite défini et modélisé sous forme d'un programme linéaire en variables réelles et bivalentes. Une étude de la complexité, nous a conduit à conclure que le problème posé est NP-difficile, puisque le problème qui se réduit à seulement deux machines avec des temps de transport constants l'est aussi.

Vu que le problème est NP-difficile et le modèle proposé est assez complexe (du fait qu'il possède un très grand nombre de variables et de contraintes) pour les problèmes de petite taille, nous avons développé des heuristiques de résolution inspirées de l'algorithme de Mc Naughton et utilisant des algorithmes de tri basés sur la règle LPT. Des bornes inférieures et supérieures sont également proposées, aussi quelques sous-problèmes du problème général sont analysés et résolus en un temps polynomial.

Dans la phase expérimentale, nous avons généré aléatoirement plusieurs instances afin d'élaborer une analyse comparative entre les heuristiques développées, en tenant compte essentiellement du nombre de fois où une heuristique donnée est meilleure par rapport aux autres, ainsi que le nombre de fois où la solution trouvée est égale à la borne inférieure (de cette façon l'optimalité de la solution est garantie). Ces différents tests ont révélé 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 problèmes de petite taille.

Cependant, comme tout travail scientifique, celui-ci n'est pas achevé. Des perspectives s'ouvrent sur de nouvelles voies de recherches pour améliorer les heuristiques proposées d'une part, développer des méta-heuristiques et/ou des méthodes exactes (de type séparation et évaluation ou programmation dynamique) d'autre part. Il est aussi intéressant d'étudier d'autres sous-problèmes avec des contraintes supplémentaires ou en considérant d'autres critères.

Bibliographie

[1] K. H. Adjallah, B. Bettayeb, I. Kacem : Ordonnancement sur machines parallèles identiques avec temps de préparation par famille : application à la gestion des tâches de maintenance préventive, Université de Technologie de Troyes(UTT), $6^{ième}$ conférence francophone de Modélisation et Simulation -MOSIM '06, 2006.

[2] F. Benhammadi, B. Bouzoui, H. Hentous, A. Saci : Nouvelles investigations pour le problème d'ordonnancement Pm $/r_i$, arborescence / $\sum w_i C_i$, Laboratoire de Robotique Centre de Développement des Technologies Avancées Baba Hasen Alger, Laboratoire d'Intelligence Artificielle Ecole Militaire Polytechnique Bordj-El -Bahri Alger, 2002.

[3] J. Berit : Scheduling parallel jobs to minimize makespan, Technische universität, Berlin, 723, 2001.

[4] M. Boudhar, H. Tchikou : Problèmes d'ordonnancement avec des machines parallèles ordonnées, les annales ROAD, laboratoire LAID3,$n1$, U.S.T.H.B, 2006.

[5] M. Boustta : Minimisation du temps de fabrication sur une machine à injection avec réglages multiples, Thèse, Université Laval Québec, 2003.

[6] P. Brucker, S. Knust : complexity results of scheduling problems, page web : http://www.mathematik.uni-osnabrueck.de/research/OR/class/, 2000.

[7] J. Carlier, P. Chrétienne, Problèmes d'ordonnancement : Modélisation / Complexité / Algorithmes, Masson, 1988.

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

[9] 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 21, 1991.

[10] C. Chu, I. Kacem, R. Mellouli, C. Sadfi : Ordonnancement sur machines parallèles avec contraintes d'indisponibilité, $6^{ème}$ conférence francophone de Modélisation et Simulation - MOSIM'06, 2006.

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

[12] C. Chu, F. Yalaoui : An efficient heuristic approch for parallel machine scheduling with job splitting and sequence dependant setup times, Université de Technologie de Troyes (UTT), 2000.

[13] M. Dell'Amico, M. Iori, S. Martello : Heuristique algorithms and scatter search for the cardinality constrained P//Cmax problem, journal of scheduling, 2004.

[14] D. Dewolf : Optimisation des flux, Université du littoral cote d'Opale, 2003.

[15] S. Dunstall, A. Wirth : Heuristic methods for the identical parallel machine flowtime problem with set-up times, Computers and operations research, 32,2005.

[16] M. Dupuy : Contributions à l'analyse des systèmes industriels et aux problèmes d'ordonnancement à machines parallèles flexibles : application aux laboratoires de contrôle qualité en industrie pharmaceutique, Thèse, Centre de Génie Industriel de l'École des Mines d'Albi-Carmaux, 2005.

[17] C. Flipo-Dhaenens : Optimisation d'un réseau de production et de distribution, Thèse, Institut National Polytechnique de Grenoble, 1998.

[18] P. Galinier, M. Habib, J. K. Hao : Métha-heuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, Revue d'Intelligence Artificielle, 1999.

[19] H. Hentous, A. Saci : Deux méthodes exactes et polynomiales pour la résolution du problème Pm $/p_i = 1$, r_i,arborescence $/\sum w_i C_i$ et Pm $/r_i$, arborescence, prmp / $\sum w_i C_i$, $2^{ième}$ Conférence sur le Génie Electrique (CGE'02), Bordj-El-Bahri, Algérie, 17-18, 2002.

[20] P. Lacomme, C. Prins et M. Sevaux : Algorithmes de graphes, $2^{ième}$ édition, Groupe Eyrolles, 1994, 2003.

[21] P. Lopez, F. Roubellat : Ordonnancement de la production, Hermes science publication, 2001.

[22] I. N. Lushchakova : Two machine preemptive scheduling problem with release date, equal processing times and precedence constraints, European Journal of Operational Research, 171 : 107-122, 2006.

[23] L. Min et C. Wu : A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines, Artificial intelligence in ingineering,13 : 399-403, 1999.

[24] E. Mokotoff : An exact algorithm for the identical parallel machine scheduling problem, European Journal of Operational Research, 152 : 758-769, 2002.

[25] E. Neron, G. Riotteau, O. Scaloni : Ordonnancement sur des machines parallèles identiques avec splitting et temps de préparation dépendant de la séquence, $1^{ière}$ conférence francophone de Modélisation et Simulation -MOSIM '01, 2001.

[26] N. E. H. Saadani : Ordonnancement des systèmes de production sans temps d'arrêt machines, thèse, faculté des sciences de Tunis, 2003.

[27] C. Sadfi : Problèmes d'ordonnancements avec minimisation des encours, thèse, Institut national polytechnique de Grenoble, 2002.

[28] M. Sakarovitch : Optimisation Combinatoire, Herman, Paris, 1984.

[29] M. Sevaux, P. Thomin : Recherche tabou améliorée pour l'ordonnancement sur machines parallèles, Université de Valenciennes-Le Mont Houy, $3^{ième}$ conférence francophone de Modélisation et Simulation -MOSIM '01, 2001.

[30] É.D. Taillard : Principes d'implémentation des métaheuristiques, Métaheuristiques et outils nouveaux en R. O.

[31] Wikipédia : Encyclopédie libre Wikipédia publiée sous licence GNU FDL,2007.