N°d'ordre : 04/2007 - 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 والتكنولوجيا

Faculté de Mathématiques

MÉMOIRE

Présenté pour obtenir le diplôme de magister

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

Par

Nadjat MEZIANI

Thème :

ORDONNANCEMENT SUR MACHINES SPECIALISEES

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

M. AIT HADDADENE Hacène, Professeur, USTHB,	Président
M. BOUDHAR Mourad, Maître de Conférences, USTHB,	Directeur de thèse
M. BERRACHEDI Abdelhafid, Professeur, USTHB,	Examinateur
M. OUAFI Rachid, Maître de Conférences, USTHB,	Examinateur
M. OULAMARA Ammar, Maître de conférences, EMNancy,	Examinateur

Dédicaces

Je dédie ce modeste travail

A mon très cher père,

A ma très chère, tendre et douce mère qui a tant donné pour moi,

A ma très chère soeur Hayet et son mari Madjid,

A mes très chères soeurs : Souad, Nora, Nawel et Amel,

A mon gracieux frère Djamel,

A mon adorable petit neveu Malek,

A toute ma famille,

A Lynda et Amina,

A tous(tes) mes amis(es).

Remerciements

Je tiens tout d'abord à remercier Monsieur AIT HADDADENE Hacène, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene, pour l'honneur qu'il me fait de présider ce jury.

Je tiens à exprimer ma reconnaissance à Monsieur BOUDHAR Mourad, Maître de Conférences à l'Université des Sciences et de la Technologie Houari Boumediene, de m'avoir dirigé dans ce mémoire, pour son aide dans toutes mes démarches, ses perpétuels encouragements et sa grande patience. Qu'il trouve ici l'expression de ma profonde gratitude.

Je tiens à exprimer mes remerciements à Monsieur BERRACHEDI Abdelhafid, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene, pour s'être intéressé à ce travail et d'avoir accepté d'être parmi les membres du jury.

Mes remerciements vont également à Monsieur OUAFI Rachid, Maître de Conférences à l'Université des Sciences et de la Technologie Houari Boumediene, pour l'intérêt qu'il a accordé à mon travail et pour avoir accepté d'être examinateur de ce mémoire.

Mes sincères remerciements vont également à Monsieur OULAMARA Ammar, Maître de conférences à l'Ecole des Mines de Nancy, d'avoir bien voulu examiner ce travail et pour l'honneur qu'il me fait de participer à ce jury.

Résumé : Nous présentons deux problèmes d'ordonnancement de type flow shop dont le but est la minimisation de la date de fin de traitement des tâches (makespan). Le premier problème est l'ordonnancement sur deux machines avec recirculation de tâches sur la seconde machine. Nous montrons que ce problème est polynomial et nous donnons un algorithme pour sa résolution. Le second consiste en ordonnancement sur deux étages tels que le premier étage comprend une seule machine et le second contient deux machines identiques sur lesquelles les tâches peuvent recirculer un nombre fini de fois. Nous montrons que le problème est NP-difficile. Des heuristiques sont présentées avec des expérimentations numériques.

Mots clés : Ordonnancement, ordonnancement sur deux machines, flowshop hybrid, complexité, recirculation, heuristiques, date de fin de traitement.

Abstract : We present two scheduling problems which the aim is to minimize the makespan. The first problem is the problem of scheduling on two machines with recirculation of tasks at the last machine. We prove that the problem is polynomial and we give algorithm for the resolution. The second problem consist in the problem of scheduling at two stages such as the first stage contain one machine and the second one contain two identical machines which tasks can recirculate a finite number of times. We prove that the problem is NP-hard. Heuristics are presented with numerical experimentations.

Keywords : Scheduling, two machines scheduling, hybrid flowshop, complexity, recirculation, heuristics, makespan.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Dans le contexte de compétition internationale actuel, l'efficacité, c'est-à-dire la capacité de produire un maximum de résultats pour un minimum d'efforts fournis, est devenue une donnée critique de l'évaluation de la "santée" d'une entreprise, notamment industrielle. L'organisation de la production figure incontestablement parmi les paramètres les plus importants qui influencent la performance réalisée.

Actuellement, toute entreprise se doit d'être plus performante et d'avoir un niveau supérieur, sur le plan technique et économique, aux entreprises concurrentes. L'amélioration de la gestion de production est la base de toute recherche dans le domaine de la productivité à accroître. La productivité est la clef de la survie et du développement des entreprises industrielles.

Dans un tel contexte, pour réussir l'entreprise doit respecter essentiellement deux dimensions, à savoir une dimension technologique, qui vise à développer les performances intrinsèques des produits mis sur le marché afin de satisfaire aux exigences de qualité et de réduction du coût de possession des produits. Une dimension organisationnelle, qui vise à développer la performance en termes de durée des cycles de fabrication, respect des dates de livraison prévues.

Pour cela, il faut disposer de méthodes et d'outils de plus en plus performants pour l'organisation et la conduite de la production. Cette organisation de la production doit être vue au niveau de l'entreprise elle-même, mais aussi au niveau de la chaîne logistique dont elle constitue l'un des maillons. Pour atteindre ces objectifs, l'organisation repose en général sur la mise en œuvre d'un certain nombre de fonctions assurées par des méthodes d'autant plus sophistiquées que les systèmes d'information sont de plus en plus fiables et dynamiques. Parmi ces fonctions, l'ordonnancement joue un rôle essentiel.

Dans la terminologie de la recherche opérationnelle, un problème d'ordonnancement désigne tout problème dans lequel l'objectif est l'allocation de ressources au cours du temps, de façon à réaliser un ensemble d'activités. Des typologies plus précises complètent cette défnition très générale. De fait, la notion de problème d'ordonnancement regroupe une grande variété de problèmes différents selon la nature des opérations à réaliser (morcelables ou non, répétitives, ...), les caractéristiques des ressources (consommables ou renouvelables, interchangeables, ...), les contraintes portant sur les opérations (dates de disponibilité, précédence, ...) et les critères

à optimiser (encours, productivité, retard, ...).

La théorie de l'ordonnancement est une branche de la recherche opérationnelle. Elle consiste en la recherche de modèles mathématiques et la mise au point de méthodes de résolution efficaces des problèmes proposés. L'importance économique considérable de ces problèmes a suscité depuis plusieurs années une recherche scientifique internationale intensive. Les problèmes traités par la théorie de l'ordonnancement sont classés en différentes catégories : les problèmes de machine simple, les problèmes de machines en parallèle, les problèmes de type flow shop, job shop, open shop et l'ordonnancement de projets. Les critères à optimiser sont généralement la minimisation du temps de terminaison ou celui du retard.

Les travaux qui s'intéressent aux ateliers en ligne et plus précisement à ceux dont la gamme de fabrication est identique (flowshop) sont particulièrement nombreux dans la litérature. Quand un étage comporte plusieurs machines paralèlles, identiques ou non (uniformes ou quelconque), on parle du flow shop hybride. Ce domaine est largement abordé dans la littérature depuis les années 1970. La NP-complétude a été prouvé dans [25]. Des méthodes exactes et des méthodes approchées ont été proposées. Les méthodes exactes sont principalement des procédures par séparation évaluation [11][35] et la programmation linéaire [1][24]. Les méthodes approchées sont principalement des heuristiques et des métaheuristiques.

Des métaheuristiques ont été récemment proposées et utilisées pour la résolution des problèmes d'ordonnancement dans un flowshop hybride. Les métaheuristiques proposées sont : le recuit simulé [28][42][22] et la recherche tabou [37][36].

Dans ce travail nous considérons deux problèmes d'ordonnancement de type flow shop dont le but est de minimiser le makespan. Le premier problème est l'ordonnancement de tâches sur deux machines avec leurs recirculation un nombre fini de fois sur la seconde machine. Le second consiste en ordonnancement sur deux étages tels que le premier étage comprend une seule machine et le second contient deux machines identiques sur lesquelles les tâches peuvent recirculer un nombre fini de fois. Le problème de reciculation de tâches a été abordé par Billaut et Bertel dans [7][8]. Ils ont traité le problème d'ordonnancement multicritère du flowshop hybrid avec recirculation, le problème identifié est NP-difficile.

Le second problème traité est tiré d'une application réelle. Une série de pièces, avant d'être traitées sur un ensemble de machine paralèlles subissent un test ou un contrôl de qualité et de conformité. Les pièces non conformes ou de mauvaise qualité sont rejetées.

Ce mémoire est organisé en quatre chapitres. Le premier est consacré aux notions générales sur l'ordonnancement. Le second chapitre est réservé à l'étude du problème $F2/recr/C_{max}$. On montre que ce problème est polynomial et on donne un algorithme pour sa résolution. Le

troisième chapitre est réservé au problème $FH2(1, P2)/recr/C_{max}$ avec recirculation des tâches sur le second étage. Une identification et modélisation du problème sont effectuées. On montre que le problème sans recirculation est NP-difficile pour des durées de traitement des tâches égales à 1 sur le premier étage. Avec recirculation, le problème est polynomial pour des durées de traitement des tâches égales à 1 sur le premier et le second étage. Nous donnons un algorithme de résolution dans ce cas. On propose également quatre heuristiques pour la résolution du problème considéré, pour lesquels une application informatique sous l'environnement *Delphi*7 est implémentée pour effectuer des tests. Dans le quatrième chapitre, on présente les résultats de ces tests. Pour cela on applique les heuristiques sur des instances de différentes tâches (allant de 10 à 1000 tâches) et les durées d'exécution des tâches sont générées en suivant les quatres lois : la loi uniforme, la loi normale, la loi exponentielle et la loi de poisson. On termine par une conclusion.

Chapitre 1

Généralités en ordonnancement

Les problèmes d'ordonnancement sont présents dans tous les secteurs d'activités de l'économie depuis l'informatique jusqu'à l'industrie manufacturière. C'est pour cette raison qu'ils ont fait et continuent de faire l'objet de nombreux travaux de recherche [41].

Résoudre un problème d'ordonnancement consiste à ordonnancer c'est-à-dire programmer ou planifier, dans le temps l'exécution des tâches en leur attribuant les ressources nécessaires matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des resources au cours du temps et vise à satisfaire un ou plusieurs objectifs. Plus précisément, on parle de problème d'ordonnancement lorsqu'on doit déterminer dans le temps les dates du début et de fin de traitement des tâches, alors qu'on réserve le terme de problème de séquencement au cas où l'on cherche seulement à fixer un ordre d'exécution des tâches qui peuvent être en conflit pour l'utilisation des ressources. Un ordonnancement induit nécessairement un ensemble unique de relations de séquencement. Par contre la détermination d'un séquencement des tâches recouvre toute une famille d'ordonnancements [34].

1.1 Concepts de base en ordonnancement

1.1.1 Les tâches

Une tâche est une entité élémentaire de travail localisée dans le temps par une date de début s_i et de fin C_i, dont la réalisation est caractérisée par une durée p_i (on a $C_i = s_i + p_i$).

1.1.2 Les ressources

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

On distingue plusieurs types de ressources :

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

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

- ***Ressources doublement contraintes***
  Une ressource est doublement contrainte lorsque son utilisation instantanée et sa consommation globale sont toutes les deux limitées (source d'énergie, financement, ...).

- ***Ressources disjonctifs***
  Une ressource est dite disjonctive ou (non partageable), si elle ne peut exécuter qu'une seule tâche à la fois (machine-outil, robot manipulateur).

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

1.2 Notations et critères d'optimisation

1.2.1 Notations

Soient $T = \{T_1, T_2, \ldots, T_n\}$ l'ensemble des n tâches à ordonnancer et $M = \{M_1, M_2, \ldots, M_m\}$ l'ensemble des m machines [43].

m	nombre de machines
n	nombre de tâches
i	indice de la tâche
j	indice de machine
$p_{i,j}$	durée de l'opération sur la machine M_j de la tâche T_i (processing time)
r_i	date minimale de début de la tâche T_i (release date)
d_i	date d'achèvement souhaitée de la tâche T_i (due date)
w_i	poids de la tâche T_i(weight)
C_i	date de fin d'exécution de la tâche T_i (completion time)
F_i	temps de présence dans l'atelier ou durée de flot (flow time) de la tâche T_i : $F_i = C_i - r_i$
L_i	écart par rapport à la fin souhaitée ou retard algébrique (lateness) de la tâche T_i : $L_i = C_i - d_i$
D_i	retard vrai de la tâche T_i (tardiness) : $D_i = max(C_i - d_i; 0)$
U_i	Indicateur de retard de la tâche T_i : $U_i = 1$ si $D_i > 0,\ U_i = 0$ sinon

Les données d'un problème d'ordonnancement sont les tâches et leurs caractéristiques, les contraintes potentielles, les ressources et la fonction économique. Souvent une tâche T_i ne peut commencer son exécution avant une date de disponibilité notée r_i et doit être achevée avant une date échue d_i . Les dates réelles de début et de fin d'exécution de la tâche T_i sont respectivement s_i et C_i. Les tâches sont souvent liées entre eux par des relations d'antérioritée ; si ce n'est pas le cas, on dit qu'ils sont indépendantes. Dans certains cas, une tâche, dite morcelable ou encore préemptive, peut être exécutée par morceaux.

1.2.2 Critères d'optimisation

Trouver un ordonnancement admissible revient à déterminer une séquence de passage des opérations sur les machines respectant les contraintes du problème. Les facteurs importants dans l'évaluation d'un problème d'ordonnancement sont l'utilisation efficace des ressources, le délai global et le respect des contraintes du problème. Les variables intervenant le plus souvent dans l'expression de la fonction économique sont [43] :

- la date C_i de fin d'exécution de la tâche T_i ;
- les retards $L_i = C_i - d_i$ et $D_i = max(C_i - d_i;\ 0)$ de la tâche T_i ;
- l'indicateur de retard de la tâche T_i : $U_i = 0 \quad \text{si} \quad C_i \leq d_i,\ U_i = 1$ sinon.

Les critères, encore appelés fonctions économiques, les plus utilisés font intervenir la durée totale, le coût des stocks des encours et les retards.

1.2.3 Les problèmes d'ordonnancement

Les problèmes d'ordonnancement sont très différents d'un atelier à l'autre, et il n'existe pas de méthode universelle permettant de résoudre efficacement tous les cas. Une classification peut s'opérer selon le nombre des machines et l'ordre d'utilisation des machines pour fabriquer un produit qui dépend de la nature de l'atelier. Un atelier se définit par le nombre de machines qu'il contient et par son type. Les différents types possibles sont les suivants :

- Une machine : chaque tâche est constituée d'une seule opération.
- Machines parallèles : elles remplissent, a priori, toutes les mêmes fonctions. Selon leur vitesse d'exécution, on distingue :

  1. machines identiques (P) : la vitesse d'exécution est la même pour toutes les machines M_j et pour toutes les tâches T_i.
  2. machines uniformes (Q) : chaque machine M_j a une vitesse d'exécution propre et constante. La vitesse d'exécution est la même pour toutes les tâches T_i d'une même machine M_j.
  3. machines indépendantes (R) : la vitesse d'exécution est différente pour chaque machine M_j et pour chaque tâche T_i.

- Machines dédiées : elles sont spécialisées à l'exécution de certaines opérations. Dans cette catégorie, chaque tâche est constituée de plusieurs opérations. En fonction du mode de passage des opérations sur les différentes machines, trois ateliers spécialisés sont différenciés, à savoir :

  1. Flow shop (F) : c'est un cas particulier du problème d'ordonnancement d'atelier pour lequel le cheminement des tâches est unique : les n tâches utilisent les m machines dans le même ordre. Cet ordre, ou gamme, unique est une donné du problème. Cette particularité se rencontre très fréquemment en pratique, elle correspond par exemple à une chaîne de traitement ou de montage.

     - Fow shop de permutation : dans ce cas du flow shop simplifié, la séquence des tâches visitant une machine est la même pour toutes les machines.

     - Flow shop hybride : il s'agit d'une généralisation des problèmes de flow shop et de machines parallèles. L'atelier est constitué d'un certain nombre d'étages en série, chaque étage est constitué de plusieurs machines en parallèles.

  2. Job shop (J) : les n tâches doivent être exécutées sur les m machines, sous des hypothèses identiques à celles du flow shop, la seule différence est que les séquences

opératoires relatives aux différentes tâches peuvent être distinctes et sont propres à chaque tâche ;

3. Open shop (O) : c'est un modèle d'atelier moins contraint que le flow shop et le job shop, car l'ordre d'exécution des opérations n'est pas fixé.

1.3.3.1 Classification

Il existe une très grande variétée de problèmes d'ordonnancement. Pour leur identification et leur classification nous adoptons la notation proposée dans [23]. Cette notation est constituée de trois champs $\alpha/\beta/\gamma$.

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

– le paramètre α_1 décrit le type des machines utilisées $\alpha_1 \in \{\emptyset,\ P,\ Q,\ R,\ F,\ J,\ O\}$ avec :
$\alpha_1 = \emptyset$: une machine ;
$\alpha_1 = P$: machines parallèles identiques ;
$\alpha_1 = Q$: machines parallèles uniformes ;
$\alpha_1 = R$: machines parallèles indépendantes ;
$\alpha_1 = F$: Flow shop ;
$\alpha_1 = J$: Job shop ;
$\alpha_1 = O$: Open shop.

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

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

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

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

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

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

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

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

– le paramètre β_7 indique si un temps d'attente entre les différentes opérations d'une tâche est autorisé $\beta_7 \in \{\emptyset,\ no-wait\}$ avec :

$\beta_7 = \emptyset$: un nombre illimité d'opérations peuvent être en attente devant une machine ;
$\beta_7 = no\text{-}wait$: les opérations doivent être exécutées les unes après les autres, aucune durée d'attente devant une machine n'est admise.

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

Dans ce qui suit, nous proposons d'enrichir la notation proposée ci-dessus par Graham et al.

1. $\alpha = \alpha_1\alpha_2((\alpha_3\alpha_4^{(l)})_{l=1}^{\alpha_2})$: décrit l'environnement des machines utilisées.
   – $\alpha_1 \in \{\emptyset,\ FH,\ O,\ J,\ F\}$;
   – α_2 : indique le nombre d'étage lorsqu'il est > 1 ;
   – $\alpha_3 \in \{\emptyset,\ P,\ Q,\ R\}$;

- α_4 : indique le nombre de machines M^l à l'étage l ;

2. $\beta = \beta_1\beta_2 \dots \beta_n$: définit l'ensemble des contraintes prises en compte, aussi bien celles concernant les travaux que celles concernant le procédé de fabrication.
   - $\beta_i \in \{\emptyset, r_i, d_i, q_i, w_i, C_i, F_i, L_i, pmtn, \dots\}$;
3. γ : indique le critère d'optimisation ($C_{max}, \sum C_i, D_{max}, \dots$).

1.3 Complexité des problèmes d'ordonnancement

L'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre. Une théorie de la complexité a été développée et permet de classer les problèmes faciles et difficiles en plusieurs classes. Parmis celles-ci on cite les classes P et NP [30][44].

1.3.1 Les classes P et NP

Pour pouvoir exposer la notion de classe de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est oui ou non. Il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil k correspondant à la fonction d'objectif f. Le problème d'ordonnancement qui vise à minimiser le makespan se transforme en problème de décision suivant : existe-t-il un ordonnancement réalisable S tel que $C_{max} \leq k$?

Il est alors possible de définir la classe P regroupant les problèmes de décision qui peuvent être résolus par des algorithmes polynomiaux. Un algorithme polynomial est défini comme un algorithme dont le temps d'exécution (temps de calcul ou complexité)est borné par $O(p(x))$, où p est un polynôme et x est la longueur d'entrée d'une instance du problème. Les algorithmes dont la complexité ne peut être bornée polynomialement sont qualifiés d'exponentiels.

La classe NP regroupe les problèmes qui peuvent être résolus en un temps polynomial par un algorithme non déterministe. Ce type d'algorithme ne fournit pas toujours la même solution à un même problème car ils se reposent sur l'utilisation des nombres (pseudo) aléatoire. C'est le cas des algorithmes réalisant des choix grâce à des heuristiques.

Pour ces algorithmes, si à chaque instruction, le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel.

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

Parmi les problèmes de la classe NP, une large classe de problèmes, les problèmes NP-complets, sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre. C'est-à-dire, s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de la classe NP. Afin de décrire cette classe d'équivalence, définissons tout d'abord la réduction polynomiale entre deux problèmes. Soient $P1$ et $P2$, deux problèmes de décision. On dit que $P1$ se réduit polynomialement à P2 (noté $P1 \propto P2$) s'il existe un algorithme de résolution de $P1$, qui fait appel à un algorithme de résolution de $P2$, et qui est polynomial lorsque la résolution de $P2$ est comptabilisé comme une opération élèmentaire [44]. On dit alors d'un problème de décision qu'il est NP-complet si tout problème de la classe NP se réduit polynomialement à lui. Les problèmes d'optimisation combinatoire dont le problème de décision associé est NP-complet sont qualifiés de NP-difficiles.

1.3.2 Prouver la NP-complétude d'un problème

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

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

Ainsi, il est nécessaire de connaître des problèmes classiques, connus pour être NP-complets. Le premier problème qui a été prouvé comme étant NP-complet dans [15], est le problème de satisfaisabilité SAT qui peut être défini comme suit : étant donné une expression booléenne de la forme d'un produit de sommes (par exemple), le problème est dit satisfaisable s'il existe une affectation en 0 et 1 de ses variables de manière à ce que l'expression booléenne prenne la valeur 1.

SAT Instance : une expression booléenne E sous forme normale conjonctive
Question : E est-elle satisfaisable ?

1.3.3 Quelques problèmes NP-complets

Nous présentons ici quelques autres problèmes NP-complets dont la NP-complétude n'a pu être démontré que grâce à l'existence d'un premier problème NP-complet. Ce sont les six problèmes de base exposés par Garey et Johnson dans [18], qui détaillent ensuite la littérature concernant la complexité des problèmes.

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

toutes les clauses ?

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

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

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

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

Le tableau ci-dessous présente quelques résultats de complexité pour des problèmes d'ordonnancement :

Problème	Complexité
BCS	O(n^3 log(n))
Machines parallèles	NP-difficile
Flow-Shop de permutation	O(nlog(n))
Flow-Shop avec délai d'attente	NP-difficile au sens fort
Job-Shop	NP-difficile
Job-Shop périodique	O(nlog(n))
Open-Shop	NP-difficile

TAB. 1.1 – **Exemple de complexité pour des problèmes d'ordonnancement**

1.3.4 Hiérarchie de complexité entre les problèmes d'ordonnancement

L'étude des relations existantes entre les différents problèmes d'ordonnancement revêt un grand intérêt, dans la mesure où cela permet d'appliquer des algorithmes de résolution connus pour certaines classes de problèmes à d'autres classes qui leurs sont réductibles. Prenons l'exemple de l'ordonnancement sur machine parallèles : il est evident que le problème $P//C_{max}$ est un cas particulier du problème $Q//C_{max}$ où toutes les machines auraient la même vitesse. Ainsi, $P//C_{max}$ est réductible à $Q//C_{max}$, qui lui est plus général et tout algorithme développé pour $Q//C_{max}$ peut être appliqué au problème $P//C_{max}$. Nous rappelons que lorsque le nombre de machines n'est pas précisée, il est considéré comme variable.

Il est intéressant de voir comment la modiffication d'un seul élèment de la classification d'un problème peut affecter sa complexité. Pour cela, nous exposons, dans la figure (fig 1.1), certaines hiérarchies existant entre différents problèmes. Pour une représentation plus complète de ces relations, il est possible de se référer à Pinedo dans [39] et Blazewicz dans [10].

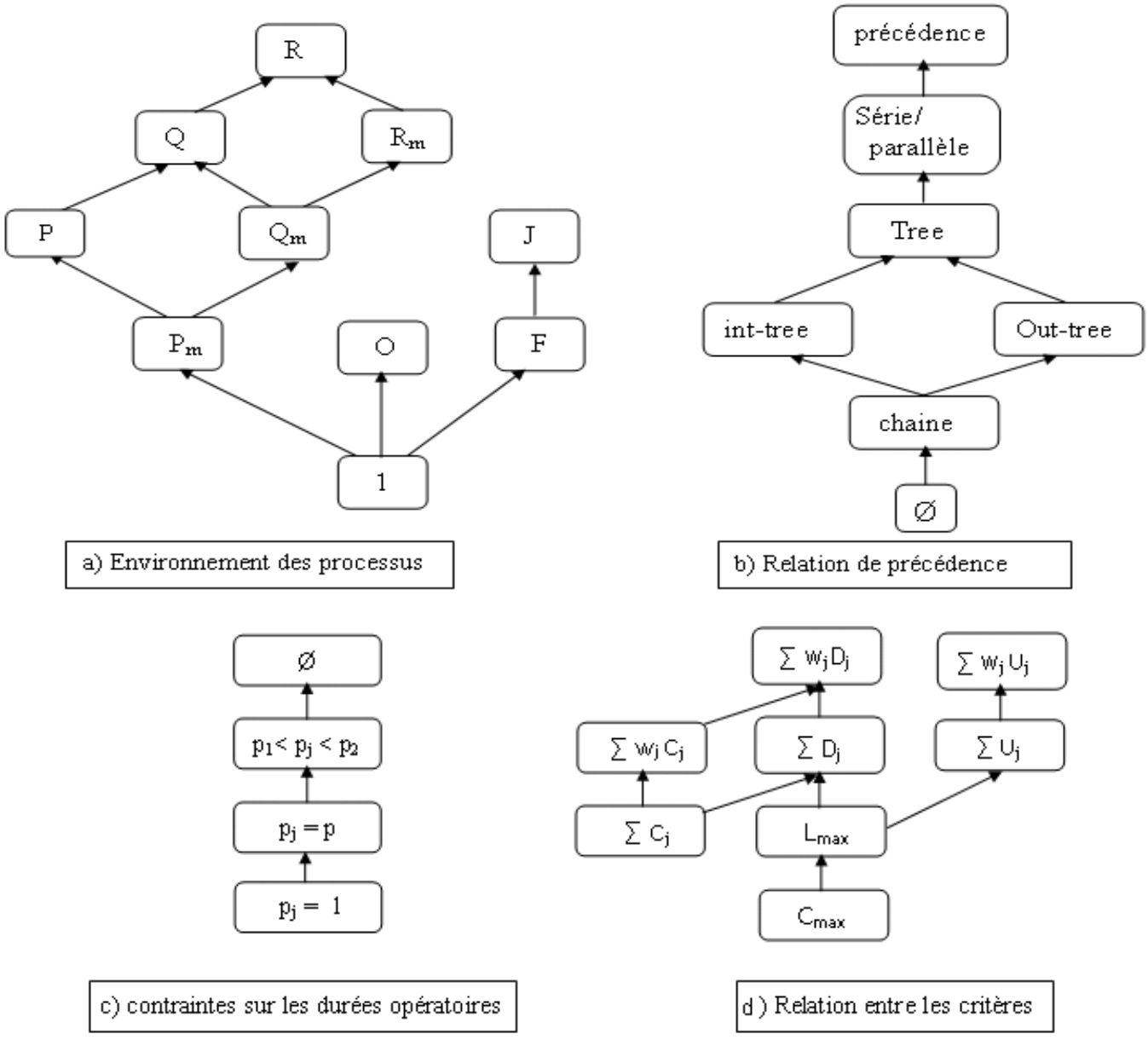

FIG. 1.1 – **Hiérarchie de complexité entre différents problèmes**

Le cas a) indique la hiérarchie de complexité entre des problèmes de configurations de machines différentes, lorsque l'environnement reste identique. Comme nous le montre la figure, le cas le plus simple est le cas à une machine. Puis le système devient plus complexe en considérant plusieurs machines en parallèle(P, Q, R) ou en série(F, O). La difficulté augmente encore, si le nombre de machines est non borné et si l'on considère, non plus des machines identiques(P), mais des machines uniformes(Q) ou différentes(R).

En ce qui concerne les relations de précédence, le cas b) nous montre clairement les degrés de complexité. Le cas le plus simple est celui dans lequel il n'existe pas des contraintes de précédende($\emptyset$), les tâches sont indépendantes. Puis le degré de complexité augmente en considérant les contraintes de précédence sous forme de chaîne et ainsi de suite jusqu'à arriver au problème le plus général et le plus complexe, le cas du problème aux contraintes générales de précédence.

La figure c) nous donne l'évolution de la difficulté, en fonction des valeurs des durées opératoires. Le problème avec des durées opératoires unitaires est un cas particulier du problème où les durées sont identiques qui lui plus général, et qui est également un cas particulier du problème où les durées opératoires des tâches sont bornées ($p_1 < p < p_2$). Ce dernier est aussi un cas particulier du problème avec des temps opératoires arbitraires ($\emptyset$) des tâches qui est le plus général des problèmes cités et le plus complexe.

Enfin, la figure d) donne la relation existant entre les principaux critères d'optimisation. En suivant le sens des flèches du schéma, chaque problème est un cas particulier d'un autre problème qui est plus complexe. Prenons l'exemple de C_{max}, ce problème est un cas particulier du problème L_{max}, ainsi C_{max} réductible à L_{max} qui lui est plus général et tout algorithme developpé pour L_{max} peut être appliqué pour C_{max}.

Si le problème le plus général est polynomial, alors le cas particulier l'est aussi et si le problème particulier est NP-difficile, donc le problème général est NP-difficile.

1.4 Etat de l'art du problème de flow shop

En matière d'ordonnancement de la production, le problème $F//C_{max}$ est le plus étudié dans la littérture. Johnson dans [30], a établi une règle qui prouve l'aspect polynomial pour un problème flow shop de permutation à deux machines. Plusieurs recherches se sont intéressées depuis à l'étude des flow shops de permutation et flow shops simple tel que les travaux de Campbell et al [12], Dannenbring [16], Taillard [45], et Widmem et Hertz [48].

Ultérieurement, des contraintes additionnelles ont été introduites, la contrainte sans délai (no-wait) exprime qu'aucune attente entre deux opérations d'une même tâche n'est permise. La contrainte de blocage exprime qu'une tâche ne doit libérée une machine que pour passer

directement sur la machine suivante et la contrainte no-idle exprime le fait qu'aucun temps mort n'est permis sur une machine entre le début de son activation et la fin de la réalisation des opérations qui lui sont affectées.

Le problème $F2/no-wait/C_{max}$ est résolu en temps polynomial par l'algorithme de Gilmore et Gomory [20].
Pour le problème du flow shop de permutation avec contrainte no-idle est peu étudié dans la littéraure, alors Baptiste et Hguny sont les seuls à avoir traité le problème $F/no-idle/C_{max}$ dans [2] et a été mentionné dans [46] comme étant NP-difficile. Le problème $F2/no-idle/C_{max}$ peut être résolu optimalement en $O(nlogn)$ par l'algorithme de Johnson.

Garey, Johnson et Sethi dans [19], ont prouvé que le problème $F3//C_{max}$ est NP-complet. Se basant sur les démonstrations qui y sont fournis, Baptiste et Hguny, dans [2] ont apporté la preuve que le problème $F3/no-idle/C_{max}$ est NP-complet au sens fort.

L'un des résultats les plus connus en ordonnancement est l'optimalité de la règle de Johnson [30] pour le problème de la minimisation du makespan dans un flow shop à deux machines ($F2//C_{max}$), qui conduit à un algorithme polynomial exact. Cette règle a servi de base à de nombreuses méthodes de résolution, qu'il sagissent d'algorithmes exacts pour des problèmes polynomiaux , d'heuristiques ou de bornes inférieures pour des problèmes NP-difficiles.

Johnson a montré que toute séquence respectant la règle $\min(p_{1i}, p_{2j}) \leq \min(p_{1j}, p_{2i}) \Longleftrightarrow (i \prec j)$ est optimale et a proposé un algorithme de complexité $O(n^2)$ puis un autre algorithme plus efficace, de complexité $O(nlogn)$ [13]. Cet algorithme est le suivant :

Algorithme. Johnson
début
- *Construire l'ensemble des tâches* $X = \{T_i / p_{1i} \leq p_{2i}\}$,
- *Ordonnancer sur les deux machines les tâches de* X *suivant l'ordre croissant de leurs temps de traitement sur la première machine,*
- *Ordonnancer sur les deux machines les tâches de* $T \setminus X$ *suivant l'ordre décroissant de leurs temps de traitement sur la deuxième machine.*

fin

Les ordonnancements de permutation sont dominants pour la minimisation de makespan dans un flow shop à deux machines. Sur la base de résultat de johnson, plusieurs auteurs se sont intéressés à déterminer un ensemble de solutions optimales pour $F2/prmu/C_{max}$.

Dans [4] est par exemple proposé un algorithme permettant d'énumérer toutes les séquences satisfaisant la règle de Johnson. Dans [6], les auteurs proposent aussi un algorithme qui énumère, en permutant des tâches des séquences optimales de Johnson, l'ensemble complet des séquences

optimales. Une approche connexe basée sur la notion de séquence maximale est aussi proposée dans [5].

Les problèmes d'ordonnancement de type flow shop hybrid sont largement étudiés depuis les années 1970. Les premiers travaux s'intéressent aussi bien à des ateliers composés de deux étages qu'à des atelies composés d'un nombre indéterminé d'étages. Ils fournissent aussi bien des méthodes approchées que des méthodes exactes. Gupta [25] montre que le problème à deux étages, avec pour objectif la minimisation de la plus grande date de fin, est NP-difficile au sens fort dès qu'un étage a plus d'une machine.

Gupta et Tunc dans [27] proposent une étude du problème $FH2(1, Pm)//C_{max}$. Ce problème est le problème inverse de celui abordé dans [25]. Les auteurs montrent que le problème est NP-difficile et fournissent deux heuristiques qui seront réutilisées dans une PSE. La première heuristique détermine dans un premier temps une liste de tâches ordonnées selon les $f(i)$ croissants [38]. Ensuite, l'affectation des tâches sur les machines du deuxième étage se fait sur la machine libre le plus tard, permettant à la tâche de ne pas attendre et, si ce n'est pas possible, sur la machine qui minimise le temps d'attente de la tâche.

Les valeurs des $f(i)$ se calculent avec la formule suivante :
$f(i) = sign(p_{1i} - p_{2i})/min(p_{1i}, p_{2i})$

$$\text{avec } sign(p_{1i} - p_{2i}) = \begin{cases} 1, & \text{si } p_{1i} > p_{2i}\,; \\ -1, & \text{si } p_{1i} < p_{2i}\,; \\ comme\ \ on\ \ veut, & \text{sinon.} \end{cases}$$

Heuristique1
début
- *Etape* 1 *: Détermination d'une séquence*
  Soit $S = (T_1, T_2, \ldots, T_n)$ *la liste des tâches classées par* $f(i)$ *croissant.*
  S'il y'a égalité, favoriser la tâche ayant la plus petite durée d'exécution au premier étage ou la plus grande durée d'exécution au deuxième étage.
- *Etape* 2 *: Affectation de la séquence obtenue :*
  Pour $p = 1$ *à* n *faire* :
  - *soit* Ω *l'ensemble des machines du deuxième étage dont la date de disponibilité est inférieure à la date de fin d'exécution de la tâche* T_p *au premier étage.*
  - *Si* Ω *est vide alors affecter la tâche* T_p *à la première machine libre.*
    Sinon affecter la tâche T_p *sur la dernière machine libre de* Ω.
    FinSi

  FinPour
- *Etape* 3 : *Retourner au problème initial.*

fin

Heuristique2

début

- *Etape* 1 *: Détermination d'une séquence*
  Soit $S = (T_1, T_2, \ldots, T_n)$ *la liste des tâches classées selon LPT.*
  S'il y'a égalité, favoriser la tâche ayant la plus petite durée d'exécution au premier étage.
- *Etape* 2 *: Affectation de la séquence obtenue :*
  Pour $p = 1$ *à* n *faire* :
  - *soit* Ω *l'ensemble des machines du deuxième étage dont la date de disponibilité est inférieure à la date de fin d'exécution de la tâche* T_p *au premier étage.*
  - *Si* Ω *est vide alors affecter la tâche* T_p *à la première machine libre.*
    Sinon affecter la tâche T_p *sur la dernière machine libre de* Ω.
    FinSi

  FinPour
- *Etape* 3 : *Retourner au problème initial.*

fin

La deuxième heuristique est basée sur [32] : il affirme qu'il est préférable d'ordonnancer les tâches, à l'étape 1, selon la règle LPT, appliquée sur les durées d'exécution du deuxième étage. Dans le cas où le nombre de machines au deuxième étage est supérieur ou égal au nombre de tâches, il suffit de classer les tâches par ordre décroissant de leur durée d'exécution au deuxième étage pour minimiser le temps d'achèvement total. Gupta et Tunc fournissent ensuite deux bornes inférieures qu'ils utiliseront dans une amélioration de la PSE de [11]. Ils proposent de prendre comme borne supérieure, en entrée de la PSE, la meilleure des deux solutions données par les deux heuristiques.

$$LB_1 = \sum_{i=1}^{n} p_{1i} + \min_{1 \leq i \leq n} p_{2i}$$

$$LB_2 = \min_{1 \leq i \leq n} p_{1i} + \left\lceil \frac{\sum_{i=1}^{n} p_{2i}}{M^{(2)}} \right\rceil$$

Plus le nombre de tâches est important, meilleurs semblent être les résultats des deux heuristiques. Les auteurs soulignent que l'heuristique $H2$ est meilleure.

1.5 Résolution des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont de complexité différente. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi,

différentes méthodes de résolution (méthodes exactes ou heuristiques), sont largement utilisées pour appréhender les problèmes NP-difficiles. Nous exposons dans cette partie, les grandes lignes des méthodes les plus connues classées en trois catégories : les méthodes exactes, les heuristiques constructives et les méthodes amélioratives. Aucune de ces méthodes n'est appliquée sur le problème étudié dans le chapitre 3, mais qui sera l'un de nos objectifs plus tard.

1.5.1 Méthodes exactes

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

1.5.1.1 Programmation dynamique

Introduite par Bellman dans les années 50 [3], la programmation dynamique se compose d'un problème de dimension n en n problèmes de dimension 1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se faisant à partir des lois d'évolution du système et d'une décision [14].

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

1.5.1.2 La méthode de séparation et évaluation

Est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Pour cela, la séparation consiste à décomposer le problème initial en plusieurs sous-problèmes qui sont à leur tour décomposables. Ce processus peut se visualiser sous forme d'un arbre d'énumération. Pour chaque sous problème (noeud de l'arbre), la procédure d'évaluation calcule (dans le cas d'un problème de minimisation) une borne inférieure de la solution obtenue à partir de ce sous problème. Au préalable, une borne supérieure de la solution optimale a été calculée et est utilisée pour éviter l'exploration de noeuds dont la valeur de la borne inférieure est

supérieure à la valeur de la borne supérieure. Cette borne supérieure est réactualisée lorsqu'une solution réalisable de valeur inférieure est atteinte. Ainsi, l'exploration de certaines branches de l'arbre est coupée, ce qui permet de ne pas énumérer réellement toutes les solutions. Il faut donc remarquer que l'efficacité d'un algorithme de séparation et d'évaluation est déterminée par la qualité des bornes utilisées et par de bonnes conditions de branchement.

1.5.1.3 Modélisation analytique et résolution

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

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

1.5.2 Méthodes heuristiques constructives

Pour les problèmes de grandes tailles, les méthodes exactes ne sont pas envisageables de par leur temps de calcul. Il est dans ce cas possible d'utiliser des méthodes approximatives qui donnent des solutions certes sous optimales, mais obtenues rapidement. Ces solutions peuvent ensuite servir de solution initiale pour les méthodes amélioratrices. La performance de tels algorithmes est généralement calculée par le rapport entre la valeur de la solution calculée par l'heuristique et la valeur de la solution optimale : $R_A(I) = A(I)/OPT(I)$ pour le pire des cas. D'autres analyses de performances peuvent être menées. Par exemple, l'analyse en moyenne regarde le comportement moyen de l'heuristique en calculant $R_A(I)$, non pas seulement pour le pire des cas, mais également pour la moyenne de plusieurs instances. De plus, lorsque la solution optimale n'est pas calculable (parce que les problèmes sont de trop grande taille, par exemple), il est également possible d'étudier expérimentalement le comportement de l'heuristique en comparant ses performances soit avec les performances d'autres heuristiques, soit avec des bornes inférieures de la solution optimale.

Les méthodes par construction progressive sont des méthodes itératives où à chaque itération, une solution partielle est complétée. La plupart de ces méthodes sont des algorithmes gloutons car elles considèrent les élèments dans un certain ordre sans jamais remettre en question un

choix, une fois qu'il a été effectuée. De principe très simple, ces algorithmes permettent de trouver une solution très rapidement. Parmi ces méthodes nous en exposons deux types.

Regardons tout d'abord, les algorithmes de liste. Ces algorithmes consistent, dans une première phase, à calculer une liste qui donnera l'ordre de prise en compte des élèments. Cette liste construite a priori, à partir d'un critère bien défini, n'est pas remise en cause au cours de l'ordonnancement. La deuxième phase de l'algorithme se réduit à considérer les élèments (processeurs ou tâches) dans l'ordre de la liste pour construire l'ordonnancement.

Un deuxième type de méthode par construction progressive, consiste à choisir, au cours de la construction, l'affectation d'une tâche sur une machine en utilisant des règles de prioritée. Ces méthodes peuvent également être vues comme des méthodes sérielles dynamiques où les listes sont reconstruites à chaque étape, selon un critère qui peut évoluer dans le temps (le nombre de tâches disponibles, par exemple). Notons que si le choix de la tâche à ajouter est guidée par l'application d'un théorème de dominance, qui permet d'établir des résultats concernant l'existence de solutions optimales si certaines hypothèses sont vérifiées [30], la méthode peut donner la solution optimale.

1.5.3 Méthodes amélioratrices

Nous exposons, dans ce paragraphe, les méthodes d'améliorations locales les plus connues. Ces méthodes sont initialisées par une solution réalisable, calculée soit aléatoirement soit à l'aide d'une des heuristiques constructives exposées précédemment, et recherchent à chaque itération une amélioration de la solution courante par des modifications locales. Cet examen se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. L'utilisation de ces heuristiques itératives suppose que l'on puisse définir pour toute solution S, un voisinage de solution, $N(S)$, contenant les solutions voisines (proches dans un certain sens). En général, le voisinage d'une solution est générée en appliquant, plusieurs fois et de façon différente, à cette solution, une petite transformation (échanges, par exemple). A chaque solution S, nous associons le coût de cette solution, $c(S)$.

1.5.3.1 Méthode de descente

C'est l'une des heuristiques de recherche locale les plus simples. Elle consiste à rechercher dans le voisinage de la solution courante, une solution de coût plus faible. Elle procède ainsi jusqu'à arriver à un optimum local. Cette méthode a l'avantage d'être rapide, mais s'arrête dès qu'un optimum local est atteint, même si celui ci n'est pas de bonne qualité.
Parmi ces méthodes, nous décrivons ici les méthodes d'échanges de type r-opt. Ces méthodes d'optimisation ont été initialement proposées par Lin dans [33] pour résoudre le Problème du Voyageur de Commerce (PVC), mais elles s'appliquent également à tout problème combinatoire dont la solution consiste en une permutation de r composantes parmi n.

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

1.5.3.2 Recherche tabou

La méthode de recherche avec tabous est basée sur des mécanismes inspirés de la mémoire humaine [17]. Son principe de base est simple : la méthode tabou fonctionne avec une seule configuration courante à la fois. Au départ une seule solution quelconque qui est actualisée au cours des itérations successives. A chaque itération, le mécanisme de passage d'une configuration, soit s, à la suivante, soit t, comporte deux étapes :

- On construit l'ensemble des voisins de s c'est à dire l'ensemble des configurations accessibles en un seul mouvement élèmentaire à partir de s. Soit $v(s)$ l'ensemble de ces voisins,
- On évalue la fonction objectif f du problème en chacune des configurations appartenant à $v(s)$. La configuration t, qui succède à s dans la suite des solutions construites par la méthode tabou est la configuration de $v(s)$ en laquelle f prend la valeur minimale.
  Notons que cette configuration t est adoptée même si elle est moins bonne que s, c'est à dire si $f(t) > f(s)$: c'est grâce à cette particularité que la méthode tabou permet d'éviter les minimums locaux.

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

1.5.3.3 Recuit simulé

Elle s'inspire des méthodes de simulation de Metropolis (année 50) en mécanique statistique. La technique du recuit en métallurgie consiste à porter le matériau à haute température puis à abaisser lentement celle-ci, ce qui permet d'obtenir un état solide bien ordonné d'énergie minimale. La méthode du recuit simulé transpose le procédé du recuit à la résolution d'un problème d'optimisation : la fonction objectif du problème, analogue à l'énergie d'un matériau, est alors minimisée, moyennant l'introduction d'une température fictive qui est dans ce cas un simple paramètre de contrôle de l'algorithme [17].

1.5.3.4 Algorithmes génétiques

Cette classe de méthodes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. L'application de ces méthodes aux problèmes d'optimisation a été formalisée par Goldberg en 1989 dans [21].

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

Chapitre 2

Flow shop à deux machines avec recirculation

Dans ce chapitre, nous proposons une étude du problème d'ordonnancement $F2/recr/C_{max}$. Ce dernier est de type flowshop sur deux machines avec duplication de tâches sur la deuxième machine dont l'objectif est la minimisation de la date de fin de traitement de l'ensemble des tâches (makespan).

Nous commençons par une présentation du problème ainsi que les différentes notations nécessaires à sa définition. Un exemple illustratif est aussi donné et un algorithme pour sa résolution est proposé suivi d'un exemple d'application.

2.1 Notation et position du problème

Nous considérons le problème d'ordonnancement $F2/recr/C_{max}$ suivant :

Soit $T = \{T_1, T_2, \ldots, T_n\}$ l'ensemble de n tâches indépendantes à ordonnancer sur un ensemble de deux machines $M = \{M_1, M_2\}$. L'atelier est de type flow shop. Chaque tâche T_i doit être exécutée une première fois sur la machine M_1 et un nombre fini de fois nb_i sur la machine M_2 tel que :

- chaque opération est exécutée sur une machine sans interruption (la préemption des tâches n'est pas aurorisée),
- une machine ne peut exécuter qu'une seule opération à la fois,
- une opération ne peut être exécutée par plus d'une machine.

Le problème précédent est illustré dans la figure suivante :

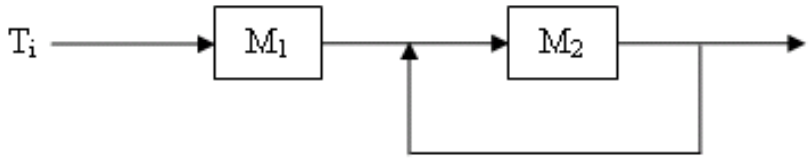

FIG. 2.1 – **Le problème à deux machines**

Pour un ordonnancement quelconque des tâches notons :
- p_{1i} : la durée d'exécution d'une tâche T_i sur la première machine ;
- $p_{2i1},\ p_{2i2}, \ldots,\ p_{2inb_i}$: les durées d'exécution d'une tâche T_i sur la deuxième machine pour la nb_i ième fois ;
- t_{ik} : date de début d'exécution d'une tâche T_i en position k sur la machine j.

L'objectif est la minimisation de la date de fin de traitement de l'ensemble des tâches.

2.2 Exemple illustratif

Pour bien illustrer le problème, considérons l'exemple suivant :
Nous disposons de 5 tâches indépendantes T_1, T_2, T_3, T_4 et T_5 à traiter sur deux machines M_1 et M_2. Les durées de traitement des tâches sur les deux machines sont données dans le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5
p_{1i}	2	3	2	4	1
nb_i	2	1	2	1	1
p_{2i1}	1	2	1	2	3
p_{2i2}	3	/	3	/	/

TAB. 2.1 – **Durées de traitement des tâches**

Les ordonnancements sont généralement représentés par des diagrammmes dits de Gantt. L'exemple donné est représenté ci-dessous. Les tâches sont traitées sur deux machines sans interruption. Chaque tâche s'exécute une seule fois sur la première machine puis un nombre fini de fois sur la deuxième machine.

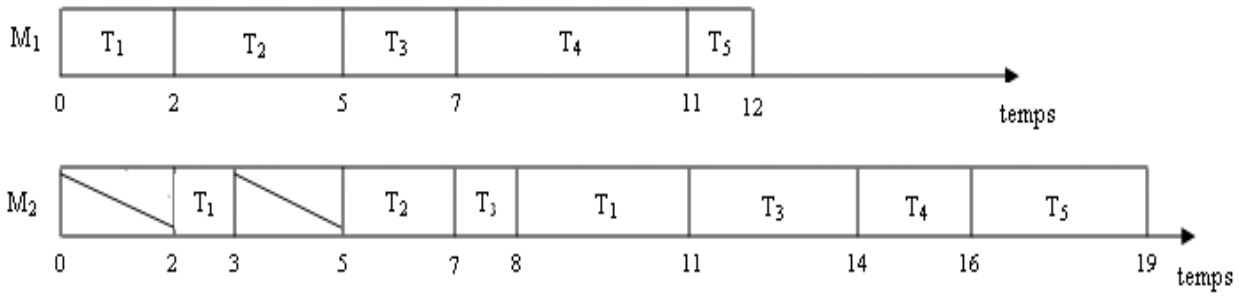

FIG. 2.2 – **Ordonnancement des tâches sur les machines M_1 et M_2**

2.3 Algorithme de résolution

L'algorithme $F2/recr/C_{max}$

début

- *Transformer le problème considéré au problème $F2//C_{max}$ comme suit :*
  - *Sur la première machine, les temps de traitement ne changent pas $p'_{1i} = p_{1i}$.*
  - *Sur la deuxième machine, les différentes opérations d'une même tâche forment une seule opération tels que : $p'_{2i} = \sum_{l=1}^{nb_i} p_{2il}$.*
- *Résoudre le problème $F2//C_{max}$ avec l'algorithme de Johnson.*
- *Construire la solution du problème initial $F2/recr/C_{max}$ en partageant la durée de traitement de chaque tâche sur la deuxième machine en durées de traitement de ses opérations élèmentaires initiales.*

fin

Exemple

Considérons 5 tâches se traitant sur deux machines M_1 et M_2 dont les durées de traitement sont données dans le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5
p_{1i}	2	3	1	2	4
nb_i	3	3	2	1	1
p_{2i1}	1	2	3	1	3
p_{2i2}	2	2	2	/	/
p_{2i3}	3	4	/	/	/

TAB. 2.2 – **Durées de traitement des tâches**

Le diagramme de Gantt suivant nous donne un ordonnancement des tâches sur les deux machines M_1 et M_2 suivant l'ordre des indices.

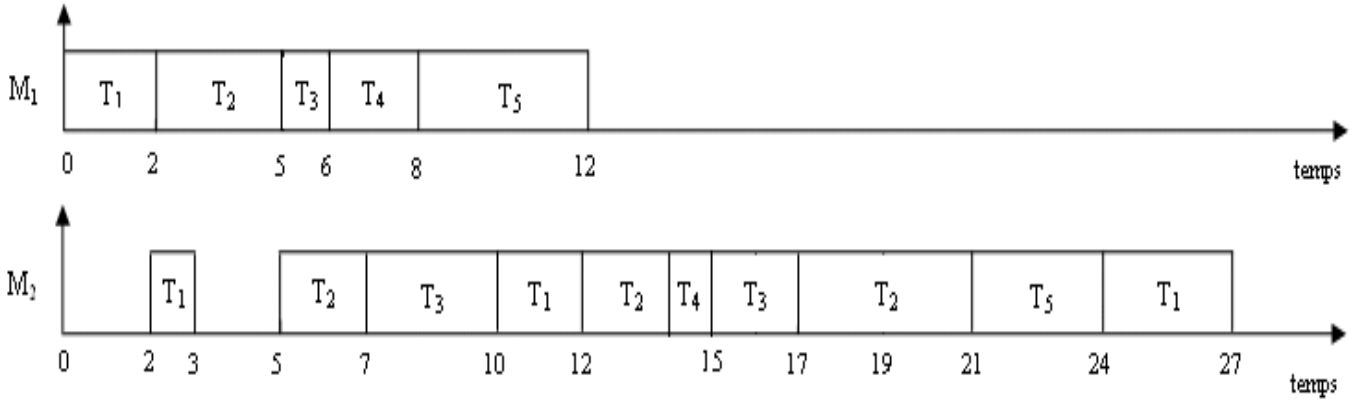

FIG. 2.3 – **Ordonnancement des tâches sur les machines M_1 et M_2**

Pour résoudre le problème de cet exemple, on applique l'algorithme $F2/recr/C_{max}$. En suivant les étapes de ce dernier on aura :

1. On transforme le problème $F2/recr/C_{max}$ au problème $F2//C_{max}$, on obtient le tableau des tâches ci-dessous en considérant le temps de traitement d'une tâche sur la deuxième machine comme étant la somme des durées de traitement de ses opérations :

T'_i	T'_1	T'_2	T'_3	T'_4	T'_5
p'_{1i}	2	3	1	2	4
$p'_{2i} = \sum_{l=1}^{nb_i} p_{2il}$	6	8	5	1	3

TAB. 2.3 – **Durées de traitement des tâches du nouveau problème**

2. En appliquant l'algorithme de Johnson, on aura l'ordonnancement donné ci-dessous :

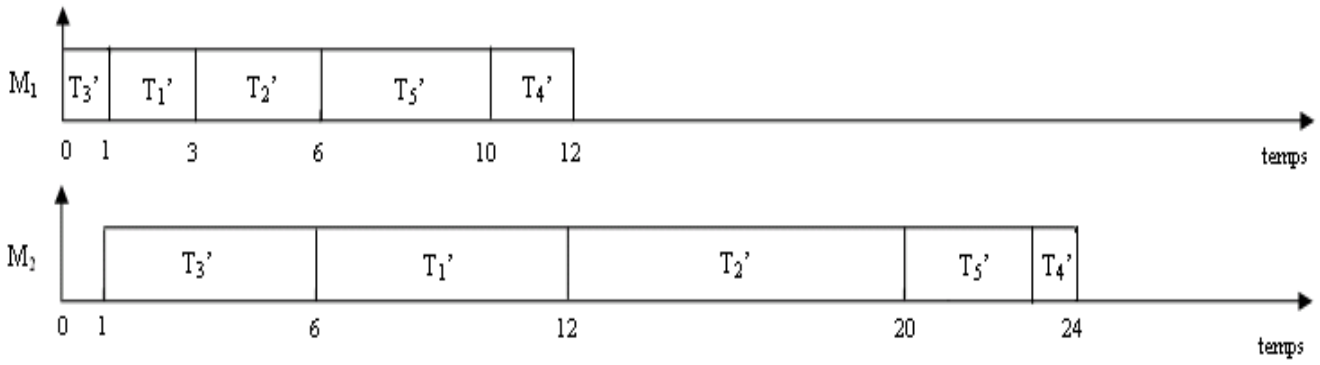

FIG. 2.4 – **Ordonnancement des tâches suivant l'algorithme de Johnson**

3. La dernière étape de l'algorithme nous donne le séquencement des opérations des tâches du problème initial avec $C_{max} = 24$. Ce séquencement est représenté dans le diagramme suivant :

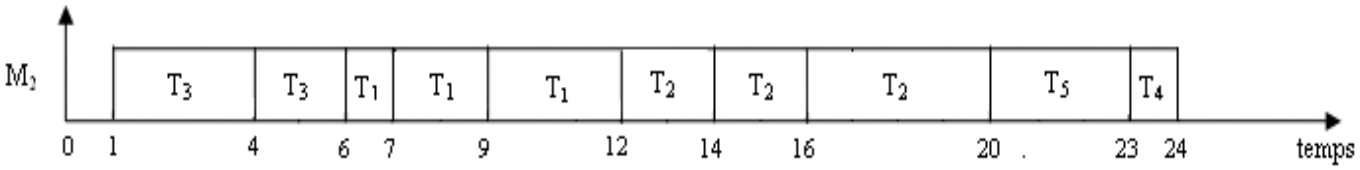

FIG. 2.5 – **Séquencement final des opérations sur** M_2

***Théorème* 2.3.1.** *L'algorithme précédent fournit une solution optimale pour le problème* $F2/recr/C_{max}$ *en* $O(nlogn + N)$ *où* $N = \sum_{i=1}^{n} nb_i$.

Pour démontrer ce théorème, on essaie de ramener le problème sur deux machines avec recirculation à un autre problème équivalent $F2//C_{max}$ pour lequel on appliquera l'algorithme de Johnson.

Preuve

Supposons que dans une solution optimale du problème $F2/recr/C_{max}$, nous avons deux opérations quelconques d'une même tâche qui ne sont pas traitées successivement (voir la figure 2.6).

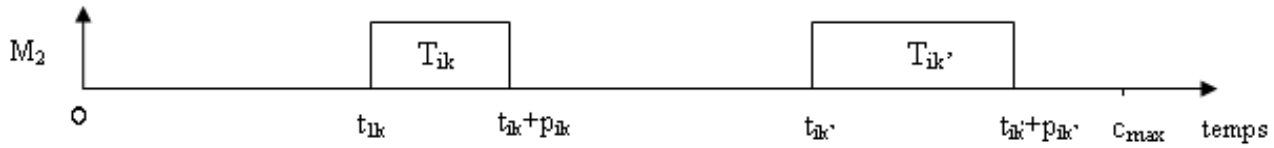

FIG. 2.6 – **Un exemple d'ordonnancement des tâches**

En traitant l'opération $T_{ik'}$ à la date $t_{ik} + p_{ik}$, la solution reste réalisable et le C_{max} ne change pas de valeur car les opérations traitées entre T_{ik} et $T_{ik'}$ seront déplacées dans le pire des cas de $p_{ik'}$ unités de temps vers la droite, donc traitées avant $t_{ik'} + p_{ik'}$.

En répétant cette opération un nombre fini de fois, on obtient une solution où toutes les opérations d'une même tâche sont traitées successivement (l'une après l'autre sans temps mort). La première étape de l'algorithme nécessite $O(N)$ opérations et l'algorithme de Johnson s'exécute en $O(nlogn)$. ■

Nous avons donc un algorithme polynomial. Le problème $F2/recr/C_{max}$ est aussi facile à résoudre.

Chapitre 3

Flow shop hybride à deux étages avec recirculation

Dans ce chapitre, nous proposons une étude du problème d'ordonnancement $FH2(1, P2)/recr/C_{max}$. Ce problème est de type flow shop hybride à deux étages tel que le premier étage contient une seule machine et le deuxième étage comprend deux machines identiques. La duplication de tâches se fait sur le second étage. L'objectif est la minimisation de la date de fin de traitement de l'ensemble des tâches.

Nous commençons par une présentation du problème ainsi que les différentes notations nécessaires à sa définition. Un exemple illustratif est aussi donné. Nous proposons également une modélisation du problème en programme linéaire en variables réelles et bivalentes et enfin des heuristiques pour sa résolution sont données.

3.1 Notation et position du problème

Le problème considéré est le suivant :

Soit $T = \{T_1, T_2, \ldots, T_n\}$ l'ensemble de n tâches indépendantes à ordonnancer sur deux étages. Le premier étage est constitué d'une seule machine M_{11} et le second étage est composé de deux machines identiques M_{21}, M_{22}. L'atelier est de type flow shop hybride à deux étages. Chaque tâche T_i doit être exécutée une première fois sur la machine M_{11} du premier étage et un nombre fini de fois nb_i sur les deux machines M_{21}, M_{22} du deuxième étage tel que :

- chaque opération est exécutée sur une machine sans interruption (la préemption des tâches n'est pas aurorisée) ;
- une machine ne peut exécuter qu'une seule opération à la fois ;
- une opération ne peut être exécutée par plus d'une machine ;
- deux opérations d'une même tâche ne peuvent s'exécuter en même temps.

Le problème précédent est illustré dans la figure suivante :

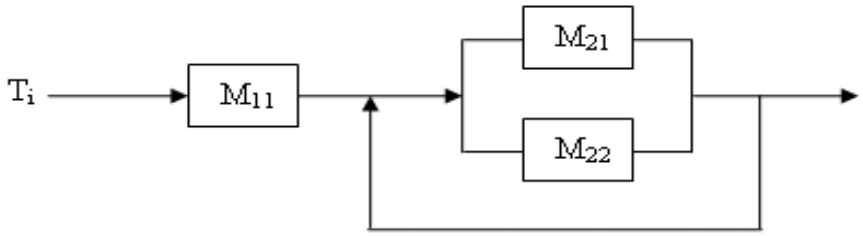

FIG. 3.1 – **Le problème à deux étages**

Pour un ordonnancement quelconque des tâches notons :
- p_{1i} : la durée d'exécution d'une tâche T_i sur la machine du premier étage ;
- p_{2ij} : la durée d'exécution de l'opération j d'une tâche T_i sur le deuxième étage ;
- s_i : la date de début d'exécution de la tâche T_i sur le premier étage ;
- r_{ijk} : la date de début d'exécution de l'opération j de la tâche T_i sur la machine k du deuxième étage.

L'objectif est la minimisation de la date de fin de traitement de l'ensemble des tâches.

Exemple illustarif

Pour illustrer le problème, considérons l'exemple suivant :
Nous disposons de 6 tâches indépendantes T_1, T_2, T_3, T_4, T_5 et T_6 à traiter sur deux étages. Les durées de traitement de ces tâches sur les deux étages sont données dans le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5	T_6
p_{1i}	2	3	2	4	1	3
nb_i	3	2	1	3	2	2
p_{2i1}	2	4	3	1	3	2
p_{2i2}	1	2	/	2	2	2
p_{2i3}	3	/	/	3	/	/

TAB. 3.1 – **Durées de traitement des tâches**

L'ordonnancement des tâches est donné dans le diagramme de Gantt suivant. Les tâches sont éxécutées une seule fois sur le premier étage et un nombre fini de fois sur le second sans interruption.

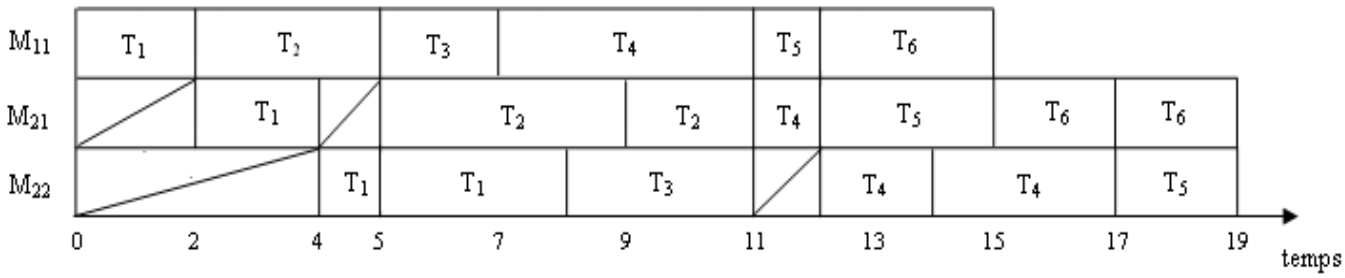

FIG. 3.2 – **Exemple d'ordonnancement des tâches sur deux étages**

3.2 Modélisation mathématique

Les variables

Soient les variables binaires α_{ij} , X_{ijk} et $\beta_{ij}^{i'j'}$ définies par :

$$\alpha_{ij} = \begin{cases} 1, & \text{si } s_i \leq s_j\,; \quad \forall\ i,j = \overline{1,n} \quad \text{et} \quad i \neq j\,; \\ 0, & \text{sinon.} \end{cases}$$

$$X_{ijk} = \begin{cases} 1, & \text{si l'opération } j \text{ de la tâche } T_i \text{ est exécutée sur la machine } k \text{ du deuxième étage}\,; \\ 0, & \text{sinon.} \end{cases}$$

pour $i = \overline{1,n}$; $j = \overline{1,nb_i}$; $k = 2,3$;

$$\beta_{ij}^{i'j'} = \begin{cases} 1, & \text{si la date de début de traitement de l'opération } j \text{ de la tâche } T_i \\ & \leq \text{à la date de début de traitement de l'opération } j' \text{ de la tâche } T_{i'}\,; \\ 0, & \text{sinon.} \end{cases}$$

pour $i, i' = \overline{1,n}$; $j, j' = \overline{1,nb_i}$;

Et soit la variable y qui correspond à la date de fin de traitement de toutes les tâches.

Les contraintes liées au premier étage

- Pour tout couple de tâche (T_i, T_j), nous avons :
  $\alpha_{ij} + \alpha_{ji} = 1\,; \qquad i,j = \overline{1,n}\,;\ i \neq j\,;$

- le traitement d'une tâche ne commence que si le traitement de la tâche qui la précède s'achève :

  $s_i + p_{1i} - s_j \leq M_1 \cdot (1 - \alpha_{ij})\,; \qquad i,j = \overline{1,n}\,; \qquad i \neq j\,;$

$s_j + p_{1j} - s_i \leq M_1 \cdot \alpha_{ij}\,; \qquad i,j = \overline{1,n}\,; \qquad i \neq j\,;$

où M_1 est une très grande valeur qui peut être égale à $\sum_{i=1}^{n} p_{1i}$.

Les contraintes liées au deuxième étage

- une opération d'une tâche n'est affectée qu'à une seule machine :
  $\sum_{k=2}^{3} X_{ijk} = 1\,; \qquad i = \overline{1,n}\,; \qquad j = \overline{1,nb_i}\,;$

- pour tout couple d'opérations nous avons :
  $\beta_{ij}^{i'j'} + \beta_{i'j'}^{ij} = 1\,; \quad i,i' = \overline{1,n}\,; \quad j = \overline{1,nb_i}\,; \quad j' = \overline{1,nb_{i'}}; \quad j \neq j' \;\; si \;\; i = i'\,;$

- sur une même machine, le traitement d'une opération d'une tâche ne commence que si le traitement de l'opération qui l'a précède est terminée :

  $r_{ijk} + p_{2ij} - r_{i'j'k} \leq M_2 \cdot (1 - \beta_{ij}^{i'j'} + 2 - X_{ijk} - X_{i'j'k})\,; \quad i,i' = \overline{1,n}\,;\; j = \overline{1,nb_i}\,;\; j' = \overline{1,nb_{i'}}\,;$
  $j \neq j' \quad \text{si} \quad i = i'\,; \quad k = 2,3\,;$

  $r_{i'j'k} + p_{2i'j'} - r_{ijk} \leq M_2 \cdot (\beta_{ij}^{i'j'} + 2 - X_{ijk} - X_{i'j'k})\,; \quad i,i' = \overline{1,n}\,;\; j = \overline{1,nb_i}\,;\; j' = \overline{1,nb_{i'}}\,;$
  $j \neq j' \quad \text{si} \quad i = i'\,; \quad k = 2,3\,;$

  où M_2 est une très grande valeur qu'on peut considérer égale à $\sum_{i=1}^{n} \sum_{j=1}^{nb_i} p_{2ij}$.

- deux opérations d'une même tâche ne peuvent pas s'exécuter au même temps sur deux machines différentes :

  $r_{ijk} + p_{2ij} - r_{ij'k'} \leq M_2 \cdot (1 - \beta_{ij}^{ij'} + 2 - X_{ijk} - X_{ij'k'})\,; \quad i = \overline{1,n}\,;\; j,j' = \overline{1,nb_i}\,;$
  $k,k' = 2,3\,;\; j \neq j'\,;\; k \neq k'\,;$

  $r_{ij'k} + p_{2ij'} - r_{ijk'} \leq M_2 \cdot (\beta_{ij}^{ij'} + 2 - X_{ij'k} - X_{ijk'})\,; \quad i = \overline{1,n}\,;\; j,j' = \overline{1,nb_i}\,;$
  $k,k' = 2,3\,;\; j \neq j'\,;\; k \neq k'\,;$

  où M_2 est une très grande valeur qu'on peut considérer égale à $\sum_{i=1}^{n} \sum_{j=1}^{nb_i} p_{2ij}$.

Les contraintes liant les dates de début de traitement des tâches du premier étage avec le deuxième étage

- si aucun ordre n'est imposé pour les opérations :
  le traitement d'une tâche ne peut commencer sur le deuxième étage que si son traitement sur le premier étage est achevé :

  $s_i + p_{1i} \leq r_{ijk}\,; \qquad i = \overline{1,n}\,; \quad j = \overline{1,nb_i}; \quad k = 2,3\,;$

la date de fin de traitement d'une tâche sur le deuxième étage est inférieure à y :
$r_{ijk} + p_{2ij} \leq y\,; \qquad i = \overline{1,n}\,; \quad j = \overline{1,nb_i}; \quad k = 2,3\,;$

– si un ordre entre les opérations est imposé :
l'exécution d'une opération d'une tâche T_i ne peut commencer que si le traitement de l'opération précédente est achevé :
$s_i + p_{1i} \leq r_{i1k}; \qquad i = \overline{1,n}; \quad k = 2,3\,;$
$r_{i1k} + p_{2i1} \leq r_{i2k}; \qquad i = \overline{1,n}; \quad k = 2,3\,;$

$\vdots$

$r_{i(nb_i-1)k} + p_{2i(nb_i-1)} \leq r_{inb_ik}; \qquad i = \overline{1,n}; \quad k = 2,3\,;$
la date de fin de traitement d'une tâche sur le deuxième étage est inférieure à y :
$r_{inb_ik} + p_{2inb_i} \leq y; \qquad i = \overline{1,n}; \quad k = 2,3\,;$

Nous remarquons que si l'opération j de la tâche T_i n'est pas traitée sur une machine k, la variable r_{ijk} peut prendre n'importe quelle valeur.

Les contraintes de positivité et de binarité

$$\begin{cases} X_{ijk} \in \{0,1\}\,; \qquad i = \overline{1,n}, \quad j = \overline{1,nb_i}, \quad k = 2,3; \\ \alpha_{ij} \in \{0,1\}\,; \qquad i,j = \overline{1,n}, \quad i \neq j; \\ \beta_{ij}^{i'j'} \in \{0,1\}\,; \qquad i\,, i' = \overline{1,n}; \quad j = \overline{1,nb_i}; \quad j' = \overline{1,nb_{i'}}; \quad j \neq j' \quad si \quad i = i'; \\ s_i \geq 0; \qquad i = \overline{1,n}; \\ r_{ijk} \geq 0; \qquad i = \overline{1,n}, \quad j = \overline{1,nb_i}, \quad k = 2,3. \end{cases}$$

La fonction objectif

$C_{max} = min(y)\,;$

Le nombre de variables et le nombre de contraintes d'un modèle mathématique linéaire sont deux indices par lesquels on peut mesurer la dimension et l'efficacité de la modélisation donnée. Le nombre de variables du modèle est donné par $(n^2 + N^2 + 3N + 1)$ et et le nombre de contraintes est égale à $\frac{5n(n-1)+N(9N-7)}{2} + 4\sum_{i=1}^{n} nb_i^2$ où $N = \sum_{i=1}^{n} nb_i$. Le but de l'expérience est d'avoir une idée sur la capacité de résoudre le problème avec le modèle. Le jugement final sur la possibilité d'application de ce modèle dépend du logiciel et des équipements informatiques disponibles.

A titre d'exemple : pour l'exemple précédent nous avons $n = 6$ et $N = 13$. Le nombre de variables est égale à 245 et le nombre de contraintes est égales à 914. Nous remarquons que pour cet exemple de petite taille, le nombre de variables et de contraintes est très important.

3.3 Le modéle final

$C_{max} = min(y)$

sous les contraintes :

$$
\begin{cases}
\alpha_{ij} + \alpha_{ji} = 1; \qquad i,j = \overline{1,n}; \quad i \neq j; \\
s_i + p_{1i} - s_j \leq M_1 \cdot (1 - \alpha_{ij}); \qquad i,j = \overline{1,n}; \quad i \neq j; \\
s_j + p_{1j} - s_i \leq M_1 \cdot \alpha_{ij}; \qquad i,j = \overline{1,n}; \quad i \neq j; \\
\sum_{k=2}^{3} X_{ijk} = 1; \qquad i = \overline{1,n}; \quad j = \overline{1,nb_i}; \\
\beta_{ij}^{i'j'} + \beta_{i'j'}^{ij} = 1; \qquad i,i' = \overline{1,n}; \quad j = \overline{1,nb_i}; \quad j' = \overline{1,nb_{i'}}; \quad j \neq j' \ \ si \ \ i = i'; \\
r_{ijk} + p_{2ij} - r_{i'j'k} \leq M_2(1 - \beta_{ij}^{i'j'} + 2 - X_{ijk} - X_{i'j'k}); \qquad i,i' = \overline{1,n}; \quad j = \overline{1,nb_i}\ ; \quad j' = \overline{1,nb_{i'}}\ ; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j \neq j' \ pour \ i = i'; \quad k = 2,3; \\
r_{i'j'k} + p_{2i'j'} - r_{ijk} \leq M_2(\beta_{ij}^{i'j'} + 2 - X_{ijk} - X_{i'j'k}); \qquad i,i' = \overline{1,n}; \quad j = \overline{1,nb_i}\ ; \quad j' = \overline{1,nb_{i'}}; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j \neq j' \ pour \ i = i'; \quad k = 2,3; \\
r_{ijk} + p_{2ij} - r_{ij'k'} \leq M_2(1 - \beta_{ij}^{ij'} + 2 - X_{ijk} - X_{ij'k'}); \qquad i = \overline{1,n}; \quad j,j' = \overline{1,nb_i}; \quad j \neq j' \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k,k' = 2,3; \quad k \neq k'; \\
r_{ij'k} + p_{2ij'} - r_{ijk'} \leq M_2(\beta_{ij}^{ij'} + 2 - X_{ij'k} - X_{ijk'}); \qquad i = \overline{1,n}; \quad j,j' = \overline{1,nb_i}; \quad j \neq j'; \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k,k' = 2,3; \quad k \neq k'; \\
s_i + p_{1i} \leq r_{ijk}; \qquad i = \overline{1,n}; \quad j = \overline{1,nb_i}; \quad k = 2,3; \\
r_{ijk} + p_{2ij} \leq y; \qquad i = \overline{1,n}; \quad j = \overline{1,nb_i}; \quad k = 2,3; \\
s_i + p_{1i} \leq r_{i1k}; \qquad i = \overline{1,n}; \quad k = 2,3; \\
r_{i1k} + p_{2i1} \leq r_{i2k}; \qquad i = \overline{1,n}; \quad k = 2,3; \\
\qquad\qquad\vdots \\
r_{i(nb_i-1)k} + p_{2i(nb_i-1)} \leq r_{inb_ik}; \qquad i = \overline{1,n}; \quad k = 2,3; \\
r_{inb_ik} + p_{2inb_i} \leq y; \qquad i = \overline{1,n}; \quad k = 2,3; \\
X_{ijk} \in \{0,1\}; \qquad i = \overline{1,n}, \quad j = \overline{1,nb_i}, \quad k = 2,3; \\
\alpha_{ij} \in \{0,1\}; \qquad i,j = \overline{1,n}, \quad i \neq j; \\
\beta_{ij}^{i'j'} \in \{0,1\}; \qquad i,i' = \overline{1,n}, \quad j = \overline{1,nb_i}; \quad j' = \overline{1,nb_{i'}}; \quad j \neq j' \ \ si \ \ i = i'; \\
s_i \geq 0; \qquad i = \overline{1,n}; \\
r_{ijk} \geq 0; \qquad i = \overline{1,n}, \quad j = \overline{1,nb_i}, \quad k = 2,3.
\end{cases}
$$

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également, parfois de le résoudre. Notre modèle est un programme linéaire en variables réelles et bivalente. L'idéal est d'obtenir un programme linéaire dont les variables sont réelles. Dans ce cas, il existe des solveurs efficaces pour le résoudre. Dès que le problème comporte des variables entières ou le modèle n'est pas linéaire, il devient plus difficile à résoudre.

Néanmoins, il est parfois surprenant de voir qu'un problème particulier de taille intéressante peut être résolu par la programmation mathématique. Il est donc justifié de commencer à étudier un problème en proposant une ou plusieurs modélisations analytiques. De plus, cette démarche a été simplifiée car il existe, actuellement, des langages de modélisation (comme MPL) permettant d'écrire des programmes linéaires de façon formelle, proche de l'écriture mathématique, et pouvant être couplés directement à un solveur comme CPLEX.

3.4 Les bornes inférieures

Dans ce qui suit nous proposons deux bornes inférieures : LB_1 et LB_2.

***Proposition* 3.4.1.** *La borne* $LB_1 = \min\limits_{1\leq i\leq n}\{p_{1i}\} + max\left\{\left\lceil \frac{1}{2}(\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{nb_i} p_{2ij} + \min\limits_{1\leq i\leq n}\{p_{1i}\})\right\rceil ; \max\limits_{1\leq i\leq n}\left\{\sum\limits_{j=1}^{nb_i} p_{2ij}\right\}\right\}$ *est une borne inférieure.*

Preuve

$max\left\{\left\lceil \frac{1}{2}\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{nb_i} p_{2ij}\right\rceil ; \max\limits_{1\leq i\leq n}\left\{\sum\limits_{j=1}^{nb_i} p_{2ij}\right\}\right\}$ est la borne inférieure de la date de fin de traitement des opérations sur le deuxième étage si leurs traitement débutent à $t = 0$. Elle est déduite du problème $P2//C_{max}$. Et bien sûr le traitement des opérations sur le deuxième étage ne peut commencer que si au moins une opération de la même tâche est achevée au premier étage. ■

***Proposition* 3.4.2.** *La borne* $LB_2 = \sum\limits_{i=1}^{n} p_{1i} + \min\limits_{1\leq i\leq n}\left\{\sum\limits_{j=1}^{nb_i} p_{2ij}\right\}$ *est une borne inférieure.*

Preuve

$\sum\limits_{i=1}^{n} p_{1i}$ est la borne inférieure de la date de fin de traitement des tâches sur le premier étage. Elle est déduite du problème $1//C_{max}$. De même le traitement des opérations sur le deuxième étage ne peut commencer que si au moins une opération de la même tâche est achevée au premier étage. ■

***Corollaire* 3.4.1.** $LB = max\{LB_1, LB_2\}$ *est également une borne inférieure.*

3.5 Etude de quelques cas particuliers

Gupta et Tunc ont montré dans [27] que le problème $FH2(1, Pm)//C_{max}$ est NP-difficile. Nous présentons une autre preuve où nous montrons que le problème reste difficile même si les temps de traitement sur la première machine sont tous égaux.

***Théorème* 3.5.1.** *Le problème $FH2(1, P2)/p_{1i} = p/C_{max}$ est NP-difficile.*

Preuve

Considérons le problème de décision connu sous le nom de 2-partition.
Etant donné n entiers positifs $a_1, a_2, \ldots, a_n$, existe t-il un sous ensemble $J \subseteq I = \{1, 2, \ldots, n\}$ / $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$? Ce problème est NP-complet [31].

Montrons que ce problème se réduit polynomialement au problème de décision 1-2-machines suivant :

Etant donné n tâches indépendantes et non morcelables. Chaque tâche a une durée de traitement qui est égale à ϵ tel que $\epsilon < \frac{1}{n}$ sur le premier étage, une durée de traitement $p_{2i} = a_i$ (une seule opération par tâche $nb_i = 1 \ \forall \ i = \overline{1, n}$) sur le deuxième étage et soit un nombre $k = \frac{1}{2} \sum_{i=1}^{n} a_i$. existe t-il un ordonnancement de durée $\leq k + 2\epsilon$?

Il est clair que cette réduction est en O(n) et que ce problème d'ordonnancement appartient à la classe NP puisqu'on peut vérifier en un temps polynomial qu'un ordonnancement donné vérifie toutes les contraintes.

Montrons que si le problème de 2-partition possède une solution, alors le problème 1-2-machines possède une solution et vice versa.

Supposons que le problème de 2-partition a une solution. Les tâches du problème d'ordonnancement sont traitées comme le montre la figure ($FIG.3.3$). Les tâches correspondantes à l'ensemble J sont traitées sur la machine M_{21} et les tâches correspondantes à l'ensemble $I \setminus J$ sur la machine M_{22}.

Le traitement d'une tâche sur le deuxième étage ne doit commencer qu'après sa fin de traitement sur le premier étage.

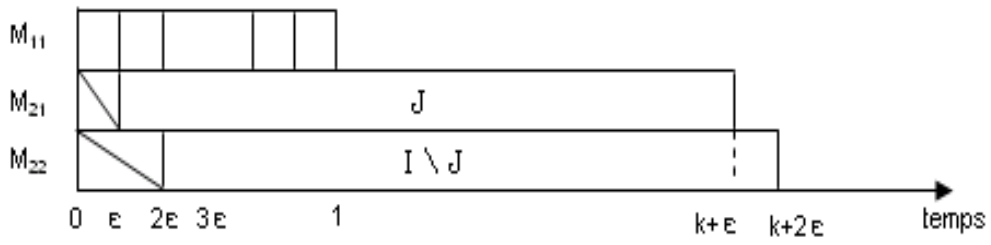

FIG. 3.3 – **Illustration du problème**

A partir de l'ordonnancement des tâches sur le deuxième étage, on peut retrouver l'ordonnancement des tâches sur le premier étage en ordonnançant en premier les tâches ayant la plus petite date de début de traitement sur le deuxième étage. Et ainsi, on ordonnance toutes les tâches sur le premier étage.
Comme la somme des temps de traitement des tâches sur chaqu'une des machines du deuxième étage est égale à k, alors $C_{max} \leq k + 2\epsilon$.
Donc le problème de 1-2-machines a une solution de durée $\leq k + 2\epsilon$.

Supposons que le problème 1-2-machines a une solution de durée $\leq k + 2\epsilon$. Donc la somme des temps de traitement des tâches traitées sur la première machine du deuxième étage est égale à k ; de même la somme des temps de traitement sur la deuxième machine du même étage est égale à k, car les temps de traitement sur les deux machines sont tous ≥ 1. Ainsi le problème de 2-partition a une solution. ■

Dans ce qui suit, nous traitons un autre cas particulier du problème, avec reciculation des tâches sur le second étage et des durées d'exécution des tâches égales à 1 sur les deux étages.
Soient : $b_1 = n + \min\limits_{1 \leq i \leq n} \{nb_i\}$, $b_2 = \left\lceil (1 + \sum\limits_{i=1}^{n} nb_i)/2 \right\rceil + 1$ et $b_3 = \max\limits_{1 \leq i \leq n} \{nb_i\} + 1$.

***Proposition* 3.5.1.** *$C^*_{max} = \max\{b_1, b_2, b_3\}$ est une borne inférieure pour le problème $FH2(1, P2)/recr, p_{1i} = 1, p_{2ij} = 1/C_{max}$.*

Preuve

C'est une conséquence directe des propositions (3.4.1) et (3.4.2) car avec $p_{2ij} = 1$ alors le temps de traitement global de chaque tâche T_i sur le deuxième étage est $\sum\limits_{j=1}^{nb_i} p_{2ij} = nb_i$. ■

Considérons l'algorithme suivant :
Algorithme (problème $FH2(1, P2)/recr, p_{1i} = 1, p_{2ij} = 1/C_{max}$) ;
début
- *calculer C^*_{max} ;*
- $t := 0$; $s := 0$;

- *pour* $i := 1$ *haut* n *faire*
  - *si il est possible de traiter la tâche* T_i *sur la première machine libre (soit* M_j*) du deuxième étage sans dépasser* C^*_{max}
    alors
    - *traiter la tâche* T_i *sur la machine du premier étage à l'instant* t *;*
    - *traiter la tâche* T_i *sur la machine* M_j *du deuxième étage sans temps mort ;*
    - $t := t + 1$ *;*

    sinon $s := i$ *;*
    fsi

  fait ;
- *si* $s \neq 0$ *alors*
  - *déplacer, d'une unité de temps, toutes les tâches du premier étage ;*
  - *déplacer, d'une unité de temps, toutes les tâches de la première machine du deuxième étage ;*
  - *déplacer, jusqu'à* C^*_{max}*, toutes les tâches de la deuxième machine du deuxième étage ;*
  - *ordonnancer la tâche* T_s *sur la machine du premier étage à* $t = 0$ *;*
  - *ordonnancer d'une unité de temps la tâche* T_s *sur la première machine du deuxième étage à* $t = 1$ *;*
  - *ordonnancer le reste de la tâche* T_s *au début de la deuxième machine (à* $t = 2$*) et à la fin de la première machine du deuxième étage si elle n'est pas entièrement traitée sur la deuxième machine ;*

  fsi

fin.

***Théorème* 3.5.2.** *L'algorithme précédent résout le problème* $FH2(1, P2)/recr, p_{1i} = 1, p_{2ij} = 1/C_{max}$ *et en* $O(n)$*.*

Preuve

L'algorithme consiste dans une première étape, à prendre les tâches, une à une, et à ordonnancer chaque tâche sans temps mort et sans dépasser C^*_{max} sur la machine du premier étage et sur la première machine libre du deuxième étage ; jusqu'à ce que, soit, toutes les tâches sont ordonnancées, soit, il reste une tâche à ordonnancer avec préemption. Un tel ordonnancement est toujours possible avec $n-1$ tâches car dans le cas de deux machines parallèles identiques, le nombre de préemptions est au plus égal à 1. Si la n-ième tâche est affectée sans préemption alors la solution obtenue est optimale. Sinon, cette dernière tâche doit être préemptée et la solution partielle obtenue se présente sous la forme de la figure suivante :

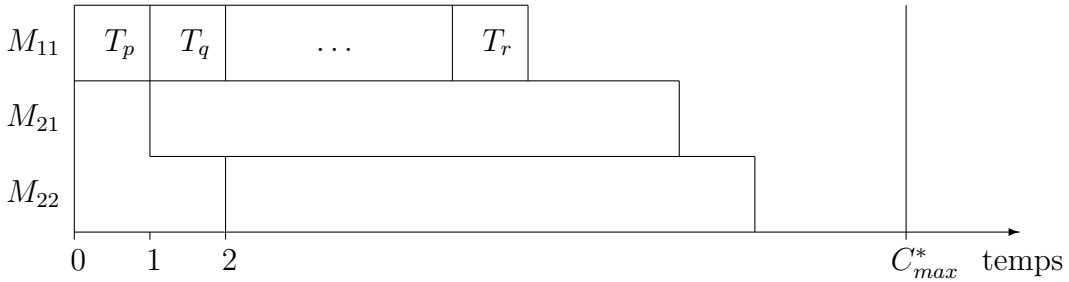

FIG. 3.4 – **Solution partielle**

Le temps de fin de traitement sur la deuxième machine du deuxième étage peut être inférieur, égal ou supérieur au temps de fin de traitement de la première machine du même étage. Il est même possible qu'aucune tâche ne soit affectée à la deuxième machine.

Alors l'ordonnancement optimal est obtenu en déplaçant, d'une unité de temps, toutes les tâches du premier étage ; en déplaçant, d'une unité de temps, toutes les tâches de la première machine du deuxième étage et en déplaçant, jusqu'à C^*_{max}, toutes les tâches de la deuxième machine du deuxième étage. Il est donné à la figure suivante où la tâche T_s est traitée dans la partie hachurée.

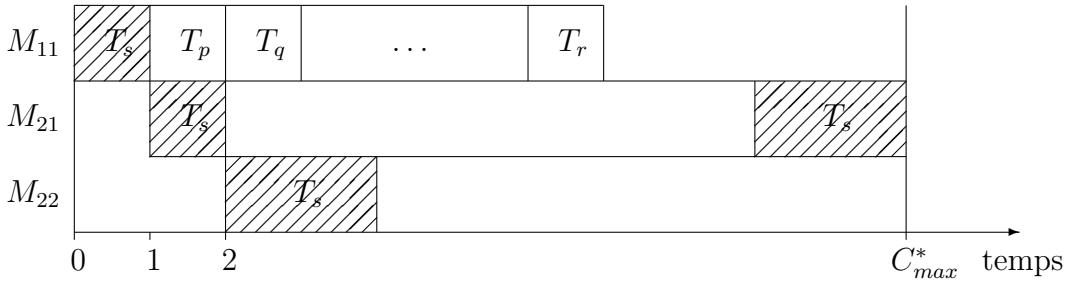

FIG. 3.5 – **Ordonnancement optimal**

exemple

Soit à ordonnancer 4 tâches de durées unitaires par opération avec les données suivantes :

T_i	T_1	T_2	T_3	T_4
nb_i	3	2	1	3

TAB. 3.2 – **Données de l'exemple**

Nous avons :
- $C^*_{max} = 6$ et $s = 4$.
- La solution optimale donnée par l'algorithme est :

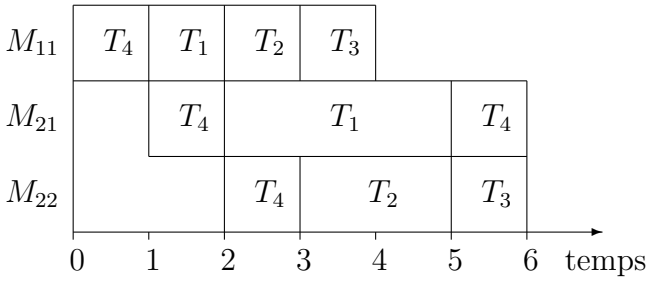

FIG. 3.6 – **Ordonnancement optimal**

On remarque que la tâche T_4 est interrompue deux fois. Une solution optimale avec un minimum de préemptions peut être obtenue en rangeant les tâches par ordre décroissant des nb_i. Elle est donnée ci-dessous.

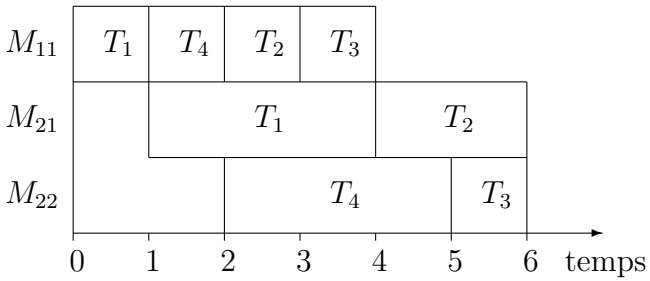

FIG. 3.7 – **Ordonnancement optimal**

3.6 Résolution du problème $FH2(1, P2)/recr/C_{max}$

Sommer les durées de traitement des opérations d'une même tâche et les traiter l'une après l'autre, sans temps mort, sur la même machine ne donne pas toujours la meilleure solution. Pour cela nous donnons un contre exemple. Alors, Considérons un ensemble de trois tâches dont les temps de traitement sont données dans le tableau suivant :

T_i	T_1	T_2	T_3
p_{1i}	1	1	1
nb_i	1	2	1
p_{2i1}	5	3	6
p_{2i2}	/	5	/
$p'_{2i} = \sum_{j=1}^{nb_i} p_{2ij}$	5	8	6

TAB. 3.3 – **Durées de traitement des tâches**

la solution optimale est :

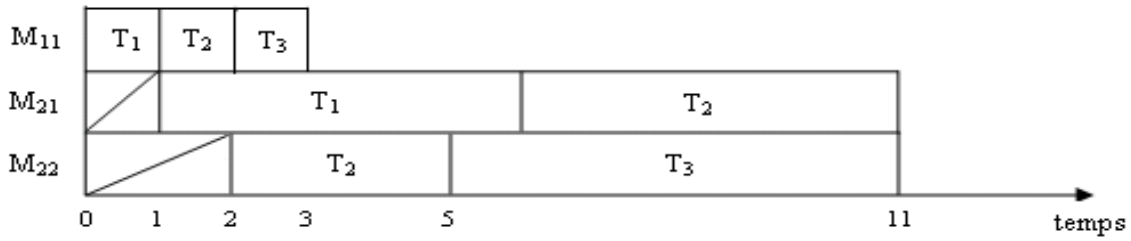

FIG. 3.8 – **Ordonnancement de tâches du contre exemple**

Nous remarquons que les 2 opérations de la tâche T_2 ne sont pas traitées l'une après l'autre, sans temps mort, sur une même machine. La solution obtenue en additionnant les temps de traitement des opérations de chaque tâche pour ne former qu'une seule tâche, et ainsi avoir le problème $FH2(1, P2)//C_{max}$, est de coût strictement supérieur.

3.7 Les heuristiques

La première étape de l'heuristique 1 se base sur l'algorithme $F2/recr/C_{max}$ qui détermine la séquence des tâches à ordonnancer sur la machine du premier étage, ensuite les tâches sont affectées sur le second étage suivant la règle FAM.

3.7.1 L'heuristique 1

Heuristique $H1$
début

- *Appliquer l'algorithme $F2/recr/C_{max}$ en ne considérant qu'une seule machine sur le second étage. On obtient un ordonnancement des tâches sur la machine du premier étage.*
- *Traiter les tâches suivant cet ordre sur la première machine.*
- *Affecter les tâches sur le deuxième étage en respectant l'ordre des tâches et leurs dates de fin de traitement sur le premier étage (algorithme de liste).*
- *Retourner au problème initial.*

fin

Gupta et Tunc [27] proposent une étude du problème $FH2(1, Pm)//C_{max}$. Les auteurs montrent que le problème est NP-difficile et fournissent deux heuristiques sur lesquelles on s'est basé pour donner trois heuristiques pour la résolution du problème $FH2(1, P2)/recr/C_{max}$.

Dans la première, on additionne dans un premier temps, les durées de traitement des opérations de chaque tâche sur le deuxième étage sur lesquelles la règle LPT est appliquée. L'affectation des tâches sur le second étage se fait sur la dernière machine libre, permettant à la tâche de ne pas attendre et si ce n'est pas le cas, sur la machine qui minimise le temps d'attente de la tâche.

3.7.2 L'heuristique 2

Heuristique *H2*

début

- *Transformer le problème* $FH2(1,P2)/recr/C_{max}$ *au problème* $FH2(1,P2)//C_{max}$ *en additionnant les durées de traitement des opérations sur le second étage de chaque tâche.*
- *Détermination d'une séquence :*
  Soit $S = (T_1, T_2, \ldots, T_n)$ *la liste des tâches classées selon la règle LPT suivant leurs durées de traitement sur le second étage. S'il y a égalité, favoriser la tâche ayant la plus petite durée d'exécution au premier étage.*
- *Affectation de la séquence obtenue :*
  Pour $p = 1$ *à* n *faire* :
  - *soit* Ω *l'ensemble des machines du deuxième étage dont la date de disponibilité est inférieure à la date de fin d'exécution de la tâche* T_p *au premier étage.*
  - *Si* Ω *est vide alors affecter la tâche* T_p *à la première machine libre.*
    Sinon affecter la tâche T_p *sur la dernière machine libre de* Ω.
    FinSi

  FinPour
- *Retourner au problème initial.*

fin

La deuxième heuristique se diffère de la première dans la détermination de la liste des tâches à exécuter sur le premier étage. On détermine une liste des tâches ordonnées selon les $f(i)$ croissant.

Remarque

$f(i) = sign(p'_{1i} - p'_{2i})/min(p'_{1i}, p'_{2i})$

$$\text{avec } sign(p'_{1i} - p'_{2i}) = \begin{cases} 1, & \text{si } p'_{1i} > p'_{2i}\,; \\ -1, & \text{si } p'_{1i} < p'_{2i}\,; \\ comme\ \ on\ \ veut, & \text{sinon.} \end{cases}$$

3.7.3 L'heuristique 3

Heuristique *H3*

début

- *Transformer le problème* $FH2(1,P2)/recr/C_{max}$ *au problème* $FH2(1,P2)//C_{max}$ *en additionnant les durées de traitement des opérations sur le second étage de chaque tâche. Calculer ensuite les* $f(i)$ *avec la formule donné ci-dessus.*
- *Détermination d'une séquence :*
  Soit $S = (T_1, T_2, \ldots, T_n)$ *la liste des tâches classées par* $f(i)$ *croissant. S'il y a égalité,*

favoriser la tâche ayant la plus petite durée d'exécution au premier étage ou la plus grande durée d'exécution au deuxième étage.

- *Affectation de la séquence obtenue :*

  Pour $p = 1$ *à* n *faire* :
  - *soit* Ω *l'ensemble des machines du deuxième étage dont la date de disponibilité est inférieure à la date de fin d'exécution de la tâche* T_p *au premier étage.*
  - *Si* Ω *est vide alors affecter la tâche* T_p *à la première machine libre.*
    Sinon affecter la tâche T_p *sur la dernière machine libre de* Ω.
    FinSi

  FinPour
- *Retourner au problème initial* ;

fin

Dans la troisième heuristique, la liste des tâches à exécuter sur le premier étage est déterminée en classant les durées de traitement des opérations des tâches, sur le deuxième étage, selon la règle LPT puis on détermine le séquencement des tâches sur le premier étage selon le séquencement de leurs opérations.

3.7.4 L'heuristique 4

Heuristique *H*4

début

- *Donner la liste des opérations des tâches à traiter sur le second étage.*
- *Ordonner cette liste suivant la règle LPT. En cas d'égalité favoriser l'opération de la tâche ayant la plus petite durée d'exécution au premier étage.*
- *Déduire le séquencement des tâches sur le premier étage d'après celui des opérations.*
- *Affectation de la séquence obtenue :*

  Pour $p = 1$ *à* $\sum_{i=1}^{n} nb_i$ *faire* :
  - *soit* Ω *l'ensemble des machines du deuxième étage dont la date de disponibilité est inférieure ou égale à la date de fin d'exécution de la tâche de l'opération* O_p *au premier étage et de la date de fin de l'exécution de son opération sur le second étage.*
  - *Si* Ω *est vide alors affecter l'opération* O_p *à la première machine libre.*
    Sinon affecter l'opération O_p *sur la dernière machine libre de* Ω.
    FinSi

  FinPour

fin

Exemple

Dans l'exemple qui suit, on applique les trois heuristiques précédentes. Considérons 5 tâches se traitant sur deux étages tel que le premier étage est constitué d'une seule machine M_{11} et le second étage est constituée de deux machines M_{21} et M_{22} dont les temps de traitement sont donnés dans le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5
p_{1i}	2	3	1	2	4
nb_i	3	3	2	1	1
p_{2i1}	1	2	3	1	3
p_{2i2}	2	4	2	/	/
p_{2i3}	3	2	/	/	/

TAB. 3.4 – **Durées de traitement des tâches**

3.8 Application des heuristiques

3.8.1 Application de l'heuristique $H1$

1. On transforme le problème $FH2(1, P2)/recr/C_{max}$ au problème $F2//C_{max}$, on obtient le tableau des tâches ci-dessous en considérant le temps de traitement d'une tâche sur le deuxième étage comme étant la somme des durées de traitement de ses opérations :

T'_i	T'_1	T'_2	T'_3	T'_4	T'_5
p'_{1i}	2	3	1	2	4
$p'_{2i} = \sum_{j=1}^{nb_i} p_{2ij}$	6	8	5	1	3

TAB. 3.5 – **Durées de traitement des tâches du nouveau problème**

2. Après l'application de l'algorithme $F2/recr/C_{max}$ sur le nouveau problème, on aura le séquencement des tâches suivant sur le premier étage :

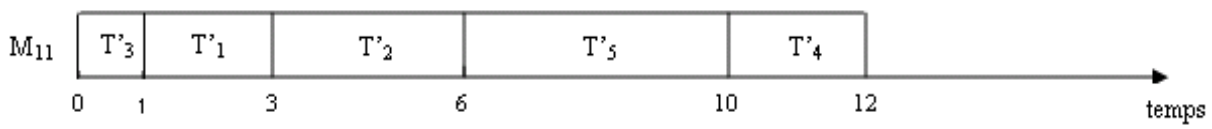

FIG. 3.9 – **Ordonnancement des tâches sur le premier étage**

3. Dans cette étape, on affecte la séquence des tâches obtenue sur le second étage, on obtient le résultat suivant :

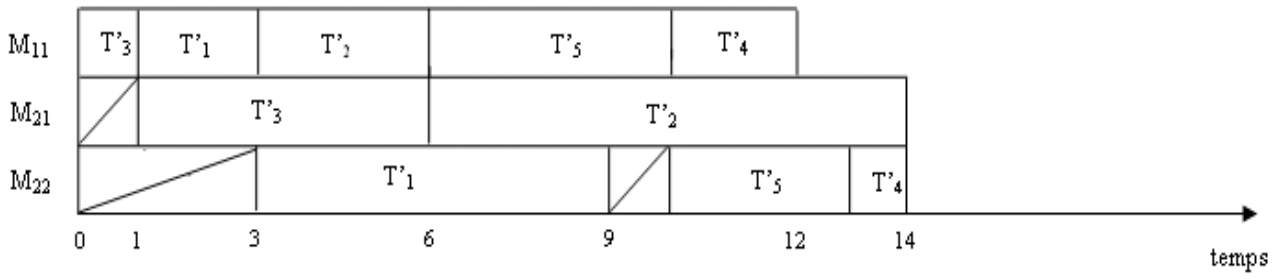

FIG. 3.10 – **Ordonnancement des tâches sur deux étages**

4. on retourne au problème initial en divisant la durée de traitement de chaque tâche en durées de traitement de ses opérations élèmentaires.

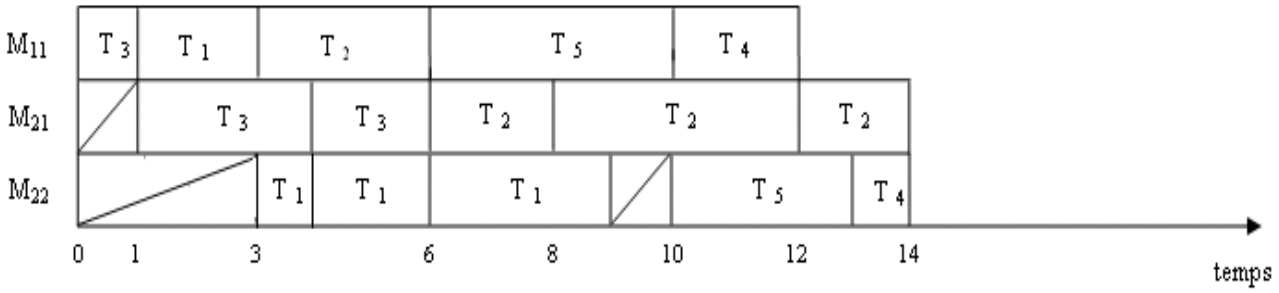

FIG. 3.11 – **Ordonnancement des tâches et opérations du problème initial**

La solution approchée donne un $C_{max} = 14$.

3.8.2 Application de l'heuristique $H2$

En suivant les étapes de l'heuristique on aura :

1. On transforme le problème $FH2(1, P2)/recr/C_{max}$ au problème $FH2(1, P2)//C_{max}$, on obtient le tableau des tâches ci-dessous en considérant le temps de traitement d'une tâche sur le deuxième étage comme étant la somme des durées de traitement de ses opérations :

T'_i	T'_1	T'_2	T'_3	T'_4	T'_5
p'_{1i}	2	3	1	2	4
$p'_{2i} = \sum_{j=1}^{nb_i} p_{2ij}$	6	8	5	1	3

TAB. 3.6 – **Durées de traitement des tâches du nouveau problème**

2. En appliquant la règle LPT sur les nouvelles tâches du deuxième étage, on aura le séquencement des tâches suivant sur le premier étage :

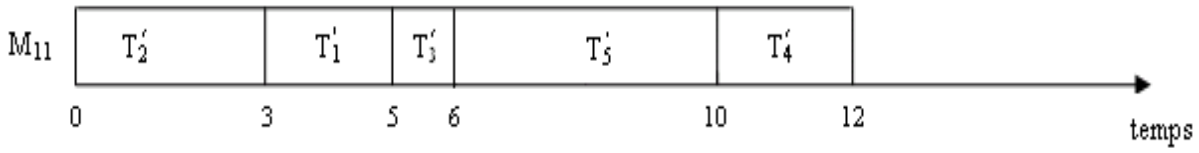

FIG. 3.12 – **Ordonnancement des tâches sur le premier étage**

3. Dans cette étape, on affecte la séquence des tâches obtenue sur le second étage, on obtient le résultat suivant :

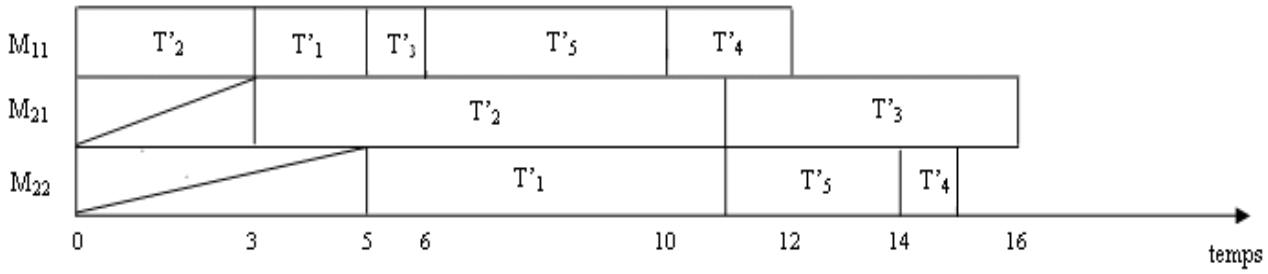

FIG. 3.13 – **Ordonnancement des tâches sur deux étages**

4. on retourne au problème initial en divisant la durée de traitement de chaque tâche en durées de traitement de ses opérations élèmentaires.

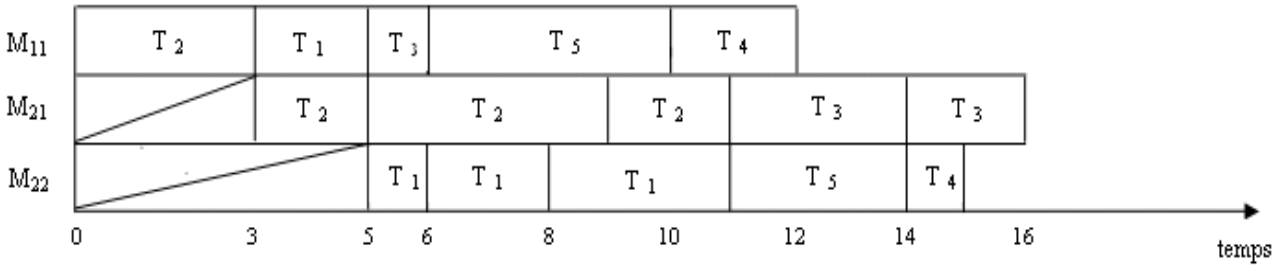

FIG. 3.14 – **Ordonnancement des tâches et opérations du problème initial**

La solution approchée donne un $C_{max} = 16$.

3.8.3 Application de l'heuristique $H3$

1. On transforme le problème $FH2(1, P2)/recr/C_{max}$ au problème $FH2(1, P2)//C_{max}$, on obtient le tableau des tâches ci-dessous en considérant le temps de traitement d'une tâche sur le deuxième étage comme étant la somme des durées de traitement de ses opérations :

T'_i	T'_1	T'_2	T'_3	T'_4	T'_5
p'_{1i}	2	3	1	2	4
$p'_{2i} = \sum_{j=1}^{nb_i} p_{2ij}$	6	8	5	1	3
$f(i)$	2	5/3	4	1	1/3

TAB. 3.7 – **Durées de traitement des tâches et leurs $f(i)$ pour le nouveau problème**

2. En classant les nouvelles tâches par l'ordre croissant de leurs $f(i)$, on aura le séquencement des tâches suivant sur le premier étage :

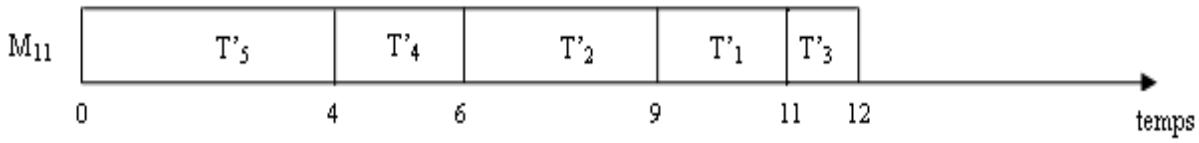

FIG. 3.15 – **Ordonnancement des tâches sur le premier étage**

3. Dans cette étape, on affecte la séquence des tâches obtenue sur le second étage, on obtient le résultat suivant :

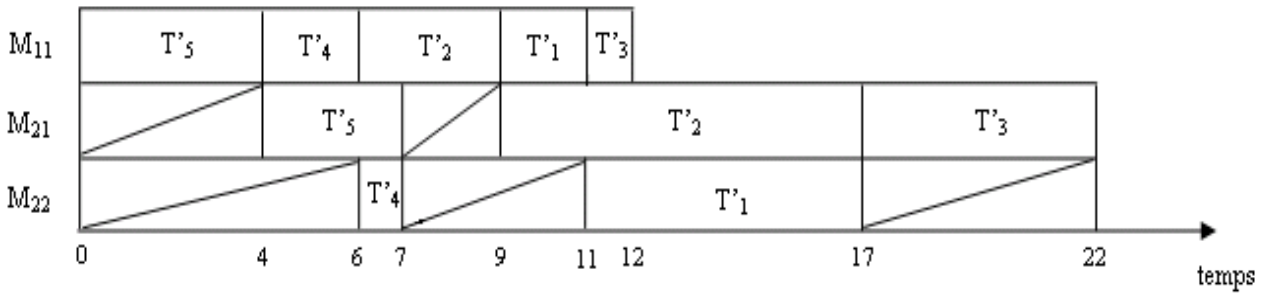

FIG. 3.16 – **Ordonnancement des tâches sur deux étages**

4. on retourne au problème initial en divisant la durée de traitement de chaque tâche en durées de traitement de ses opérations élèmentaires.

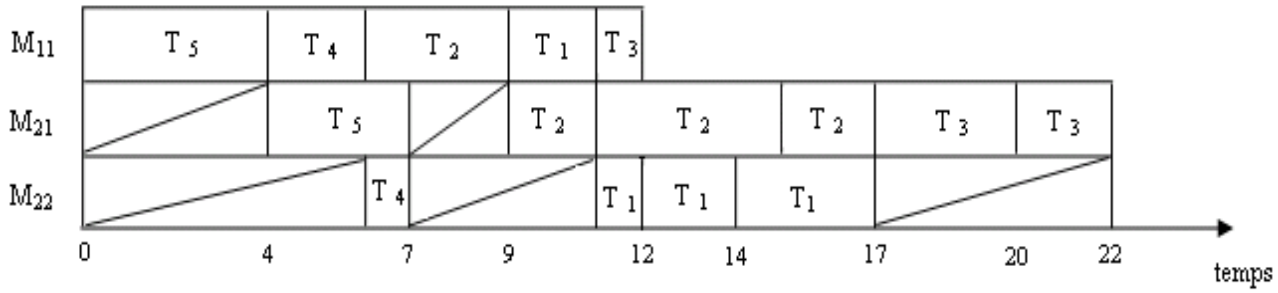

FIG. 3.17 – **Ordonnancement des tâches et opérations du problème initial**

La solution approchée donne un $C_{max} = 22$.

3.8.4 Application de l'heuristique $H4$

1. On donne la liste des opérations des tâches avec leur durées de traitement :

T_i	T_{11}	T_{12}	T_{13}	T_{21}	T_{22}	T_{23}	T_{31}	T_{32}	T_{41}	T_{51}
p_{2ij}	1	2	3	2	4	2	3	2	1	3

TAB. 3.8 – **Durées de traitement des opérations**

2. On classe les opérations suivant leurs durées de traitement selon la règle LPT :

T_i	T_{22}	T_{31}	T_{13}	T_{51}	T_{32}	T_{12}	T_{21}	T_{23}	T_{11}	T_{41}
p_{2ij}	4	3	3	3	2	2	2	2	1	1

TAB. 3.9 – **Le séquencement des opérations**

3. A partir de ce séquencement, on déduit le séquencement des tâches sur le premier étage :

T_i	T_2	T_3	T_1	T_5	T_4
p_{1i}	3	1	2	4	2

TAB. 3.10 – **Le séquencement des tâches sur le premier étage**

4. L'ordonnancement des opérations et des tâches est représenté dans la figure suivante :

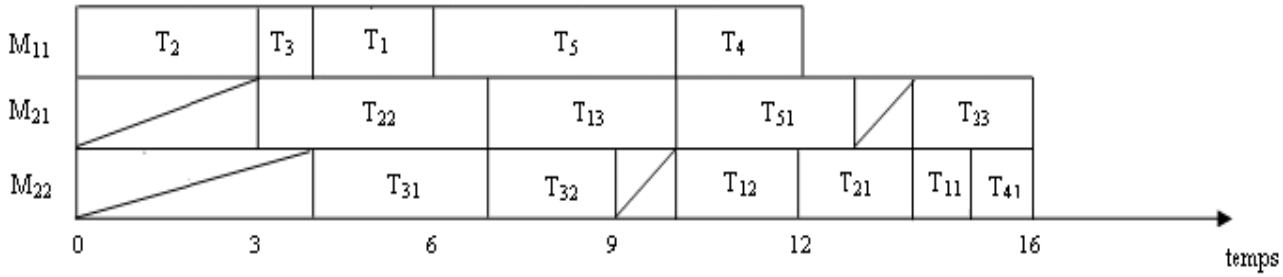

FIG. 3.18 – **Ordonnancement des tâches et opérations sur deux étages**

la solution approchée donne un $C_{max} = 16$.

Chapitre 4

Expérimentations et interprétation des résultats

Ce chapitre est consacré à la partie pratique. Pour la programmation, nous avons utilisé un micro ordinateur de type *Pentium(R)III*, de 256 *MO de RAM* et de *fréquence* de 1 *GHZ*.
Nous avons conçu une application sous l'environnement *Delphi7* sous lequel nous avons programmé les heuristiques proposés dans ce mémoire. Nous donnons quelques résultats pour les différents tests effectués pour les heuristiques du second problème. Des commentaires et remarques suivront les résultats et nous terminons par une conclusion.

4.1 Tests et résultats

Nous avons effectué des tests uniquement sur les 3 heuristiques proposées *H1*, *H2* et *H4*. L'heuristique $H3$ a donné des résultats médiocres. Pour cela, sur un nombre de tâches fixé nous avons généré les durées de traitement de ces tâches et leurs opérations suivant les lois suivantes :

1. La loi uniforme ;
2. La loi normale ;
3. La loi exponentielle ;
4. La loi de poisson.

Pour un nombre de tâches déterminé, nous avons généré 100 instances sur lesquels nous avons appliqué les heuristiques cités ci-dessus. Pour la partie résultats, nous affichons dans les tableaux qui suivent et pour chacune des 3 heuristiques, le pourcentage où la date de fin de traitement est meilleure, le nombre de fois que l'heuristique fournie une solution optimale (c'est-à-dire que la solution obtenue coincide avec la borne inférieure LB) et la durée d'exécution moyenne de chaque heuristique (en milliseconde).

4.2 Les résultats

Après l'exécution des heuristiques, on obtient les résultats présentés dans les tableaux suivants :

4.2.1 Les temps de traitement suivent une loi uniforme

		$p_i \in [1,30], nb_i \in [1,5]$			$p_i \in [1,30], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	38	32	30	38	44	18
	opt	0	0	1	0	0	1
	durée-moy	2	0.71	2	1.61	0.9	2.8
$T = 20$	C_{max}	43	40	17	21	57	22
	opt	1	1	0	0	0	1
	durée-moy	2.6	1.6	2.11	2.6	1.1	4.3
$T = 50$	C_{max}	33	55	12	25	64	11
	opt	1	0	1	2	0	0
	durée-moy	5.01	2.8	5.71	4.81	2.7	12.82
$T = 100$	C_{max}	31	60	9	10	66	24
	opt	3	0	0	1	0	0
	durée-moy	10.43	5.52	12.82	13.82	7.62	41.66
$T = 250$	C_{max}	25	70	5	12	74	17
	opt	1	0	0	1	0	0
	durée-moy	50.36	21.83	61.68	56.08	27.97	204.25
$T = 500$	C_{max}	27	68	5	10	73	17
	opt	0	1	0	0	1	0
	durée-moy	168.66	59.39	210.12	179.26	69.24	709.01
$T = 1000$	C_{max}	24	66	10	10	66	24
	opt	0	0	0	0	0	0
	durée-moy	585.16	173.97	748.2	630.81	210.72	2665.44

TAB. 4.1 – **Les résultats obtenus avec la loi uniforme**

		$p_i \in [1,50], nb_i \in [1,5]$			$p_i \in [1,50], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	40	35	25	33	52	15
	opt	0	0	1	0	0	2
	durée-moy	1.9	0.9	1.72	2.22	1.1	4.2
$T = 20$	C_{max}	43	39	18	30	53	17
	opt	0	0	2	1	0	0
	durée-moy	2.9	1.5	2.5	2.71	1.6	8.3
$T = 50$	C_{max}	39	50	11	21	65	14
	opt	0	0	0	0	0	0
	durée-moy	5.1	3	5.61	6.3	3.5	18.73
$T = 100$	C_{max}	37	56	7	14	69	17
	opt	0	1	0	0	0	0
	durée-moy	4.03	7.52	15.32	12.82	14.01	77.72
$T = 250$	C_{max}	24	73	3	12	75	13
	opt	0	1	0	0	0	0
	durée-moy	50.98	23.03	62.78	68.98	38.05	258.31
$T = 500$	C_{max}	18	76	6	10	74	16
	opt	1	0	0	0	0	0
	durée-moy	164.64	59.98	210.64	175.07	67.6	692.59
$T = 1000$	C_{max}	16	80	4	11	75	14
	opt	0	0	0	0	0	0
	durée-moy	579.87	176.16	751.57	615.76	208.31	2596.06

TAB. 4.2 – **Les résultats obtenus avec la loi uniforme**

4.2.2 Les temps de traitement suivent une loi normale

		$p_i \in [1,30], nb_i \in [1,5]$			$p_i \in [1,30], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	48	26	26	32	48	20
	opt	1	0	0	1	0	0
	durée-moy	1.4	0.6	2.6	1.4	0.6	2.2
$T = 20$	C_{max}	40	33	27	33	63	4
	opt	0	1	1	0	1	0
	durée-moy	2.3	1.2	2.41	1.9	1.4	4.61
$T = 50$	C_{max}	34	55	11	28	71	1
	opt	4	0	0	1	1	0
	durée-moy	5.32	2.21	5.9	5.21	3.31	12.3
$T = 100$	C_{max}	37	60	3	22	77	1
	opt	2	0	0	3	0	0
	durée-moy	11.72	5.9	13.34	11.7	7.21	34.46
$T = 250$	C_{max}	28	71	1	17	83	0
	opt	1	1	0	0	0	0
	durée-moy	47.57	20.63	55.89	54.56	24.31	181.08
$T = 500$	C_{max}	29	69	2	18	81	1
	opt	1	0	0	0	0	0
	durée-moy	165.25	58.79	204.09	177.18	70.01	674.98
$T = 1000$	C_{max}	30	70	0	23	77	0
	opt	0	0	0	1	1	0
	durée-moy	582.04	177.14	789.34	625.82	215.29	2512.5

TAB. 4.3 – **Les résultats obtenus avec la loi normale**

		$p_i \in [1,50], nb_i \in [1,5]$			$p_i \in [1,50], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	44	27	29	30	48	22
	opt	1	0	2	1	0	0
	durée-moy	2	1	2.4	1.8	0.5	2.5
$T = 20$	C_{max}	41	39	20	30	62	8
	opt	2	0	0	0	0	1
	durée-moy	3	2	3.6	2.32	1.3	4
$T = 50$	C_{max}	33	55	12	32	67	1
	opt	3	0	0	0	1	1
	durée-moy	4.6	2.8	7.53	5.71	2.9	12.82
$T = 100$	C_{max}	27	69	4	23	77	0
	opt	3	1	0	0	1	0
	durée-moy	12.02	6.32	14.31	11.42	6.3	33.47
$T = 250$	C_{max}	25	71	4	23	77	0
	opt	1	1	0	0	0	0
	durée-moy	48.7	21.95	59.11	52.98	24.64	187.57
$T = 500$	C_{max}	25	75	0	18	82	0
	opt	0	0	0	1	1	0
	durée-moy	166.25	60.9	207.88	174.72	68.12	670.18
$T = 1000$	C_{max}	24	76	0	16	84	0
	opt	3	0	0	0	0	0
	durée-moy	583.82	176.1	737.95	624.56	211.48	2518.21

TAB. 4.4 – **Les résultats obtenus avec la loi normale**

4.2.3 Les temps de traitement suivent une loi exponentielle

		$p_i \in [1,30], nb_i \in [1,5]$			$p_i \in [1,30], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	52	19	29	29	23	48
	opt	1	0	1	1	0	0
	durée-moy	1.5	0.8	1.3	1.8	0.8	2.68
$T = 20$	C_{max}	56	27	17	19	51	30
	opt	2	0	0	0	0	0
	durée-moy	2.3	1.2	1.71	5.4	2.5	12.93
$T = 50$	C_{max}	48	33	19	18	44	38
	opt	3	1	0	1	0	0
	durée-moy	5	3.6	5.52	5.2	3	12.32
$T = 100$	C_{max}	34	48	18	14	51	35
	opt	0	0	1	0	0	0
	durée-moy	13.12	7.2	16.13	10.72	5.3	32.86
$T = 250$	C_{max}	26	49	25	19	54	27
	opt	1	0	0	0	0	0
	durée-moy	48.16	22.52	59.2	49.8	21.72	173.38
$T = 500$	C_{max}	28	55	17	15	61	24
	opt	1	0	0	0	1	0
	durée-moy	163.12	59.31	201.69	176.54	67.8	660.61
$T = 1000$	C_{max}	26	62	12	17	71	12
	opt	0	1	0	0	1	0
	durée-moy	589.37	180.48	731.95	614.36	206.29	2510.65

TAB. 4.5 – **Les résultats obtenus avec la loi exponentielle** $\lambda = 15$

		$p_i \in [1,50], nb_i \in [1,5]$			$p_i \in [1,50], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	50	22	28	26	25	49
	opt	1	0	0	0	0	0
	durée-moy	1.5	0.7	2	3.01	1.7	6.4
$T = 20$	C_{max}	47	27	26	18	37	45
	opt	0	0	1	0	0	0
	durée-moy	2.8	1.7	2.8	3.41	1.8	9.91
$T = 50$	C_{max}	34	46	20	24	36	40
	opt	1	0	0	0	0	1
	durée-moy	4.82	2.9	6.21	8.23	6.01	24.73
$T = 100$	C_{max}	23	50	27	17	49	34
	opt	3	0	0	0	0	0
	durée-moy	10.9	5.8	13.4	20.51	14.93	68
$T = 250$	C_{max}	18	53	29	11	54	35
	opt	0	0	0	2	0	0
	durée-moy	49.2	22.21	58	67.52	38.67	248.87
$T = 500$	C_{max}	22	55	23	13	56	31
	opt	0	0	0	0	0	0
	durée-moy	166.85	61.08	206.68	193.08	84.92	783.38
$T = 1000$	C_{max}	19	64	17	16	83	1
	opt	0	0	1	0	0	0
	durée-moy	591.97	178.99	728.37	629.47	212.42	2500.54

TAB. 4.6 – **Les résultats obtenus avec la loi exponentielle** $\lambda = 25$

4.2.4 Les temps de traitement suivent une loi de poisson

		$p_i \in [1,30], nb_i \in [1,5]$			$p_i \in [1,30], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	23	70	7	19	81	0
	opt	0	0	0	1	0	0
	durée-moy	1.5	1.1	1.81	1.9	0.9	3.7
$T = 20$	C_{max}	23	73	4	14	84	2
	opt	0	1	0	0	0	0
	durée-moy	2.2	1.1	2.6	3.5	1.7	6.02
$T = 50$	C_{max}	37	63	0	22	78	0
	opt	1	0	0	0	0	0
	durée-moy	7.01	3.6	8.2	7.93	3.9	15.82
$T = 100$	C_{max}	21	79	0	15	85	0
	opt	0	0	0	1	0	0
	durée-moy	19.13	10.82	22.94	15.52	7.7	44.07
$T = 250$	C_{max}	25	75	0	14	86	0
	opt	2	0	0	0	0	0
	durée-moy	56.07	24.12	67.02	56.68	26.24	192.71
$T = 500$	C_{max}	30	70	0	18	82	0
	opt	0	0	0	0	0	0
	durée-moy	170.95	61.29	205.7	178.23	69.19	670.38
$T = 1000$	C_{max}	29	71	0	13	87	0
	opt	0	0	0	0	0	0
	durée-moy	597.56	179.17	727.71	623.96	208.75	2476.88

TAB. 4.7 – **Les résultats obtenus avec la loi poisson** $m = 15$

		$p_i \in [1,50], nb_i \in [1,5]$			$p_i \in [1,50], nb_i \in [1,10]$		
		$H1$	$H2$	$H4$	$H1$	$H2$	$H4$
$T = 10$	C_{max}	18	77	5	19	81	0
	opt	1	0	0	1	0	0
	durée-moy	2.51	0.4	2.2	2.4	0.9	3.6
$T = 20$	C_{max}	16	80	2	13	87	0
	opt	0	0	0	0	0	0
	durée-moy	3.4	1.7	3.4	5.53	1.6	9.21
$T = 50$	C_{max}	13	87	0	9	91	0
	opt	1	0	0	0	0	0
	durée-moy	6.3	3.61	7.91	6.3	2.9	16.7
$T = 100$	C_{max}	15	85	0	8	92	0
	opt	0	0	0	0	0	0
	durée-moy	14.7	8.21	17.14	15	8.92	43.66
$T = 250$	C_{max}	16	84	0	8	92	0
	opt	1	0	0	0	0	0
	durée-moy	53.68	26.64	64.18	56.47	25.14	193.56
$T = 500$	C_{max}	17	83	0	10	90	0
	opt	0	0	0	0	0	0
	durée-moy	168.97	61.19	202.99	179.62	69.89	664.88
$T = 1000$	C_{max}	26	74	0	14	86	0
	opt	0	0	0	0	0	0
	durée-moy	590.24	178.26	717.02	633.87	214.25	2491.19

TAB. 4.8 – **Les résultats obtenus avec la loi poisson** $m = 25$

4.3 Commentaires

1. La loi uniforme

- $p_i \in [1, 30]$ et $nb_i \in [1, 5]$:

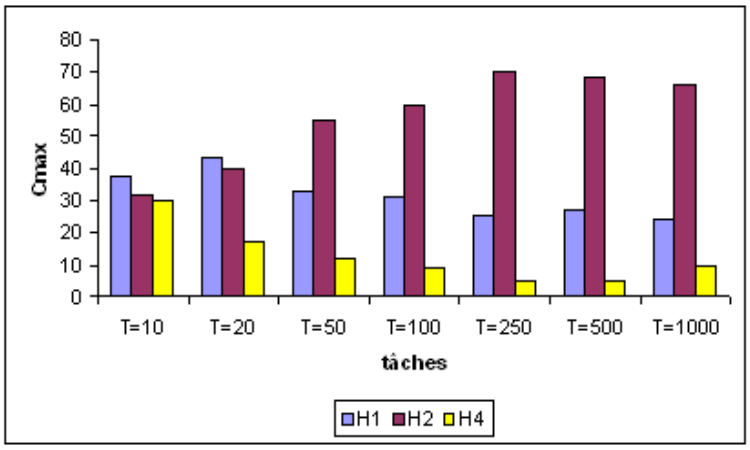

FIG. 4.1 – **Représentation des résultats de la loi uniforme par histogramme**

Nous remarquons que l'heuristique $H1$ donne une meilleure valeur de C_{max} par rapport à l'heuristique $H2$ et $H4$ pour un petit nombre de tâches ($T = 10, T = 20$). En augmentant le nombre de tâches, à partir de $T = 50$ jusqu'à $T = 1000$, l'heuristique $H2$ fournit la meilleur valeur de C_{max}, puis l'heuristique $H1$ suivie $H4$. Nous constatons que la solution ne coincide que rarement avec la borne inférieure, mais ça n'empêche pas de d'obtenir parfois des solutions optimales sauf pour un très grand nombre de tâches ($T = 1000$).

- $p_i \in [1, 30]$ et $nb_i \in [1, 10]$:

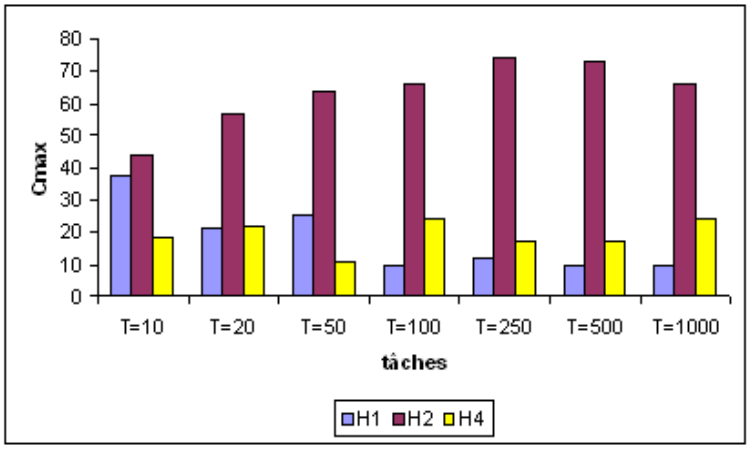

FIG. 4.2 – **Représentation des résultats de la loi uniforme par histogramme**

En augmentant le nombre d'opérations des tâches ($nb_i \in [1, 10]$), l'heuristique $H2$ devient meilleure. Les bornes inférieures sont rarement atteintes sauf pour le cas où $T = 1000$, la

borne inférieure n'est jamais atteinte, ce qui veut dire que les heuristiques ne fournissent pas de solutions optimales pour un très grand nombre de tâches.

- $p_i \in [1, 50]$ et $nb_i \in [1, 5]$:

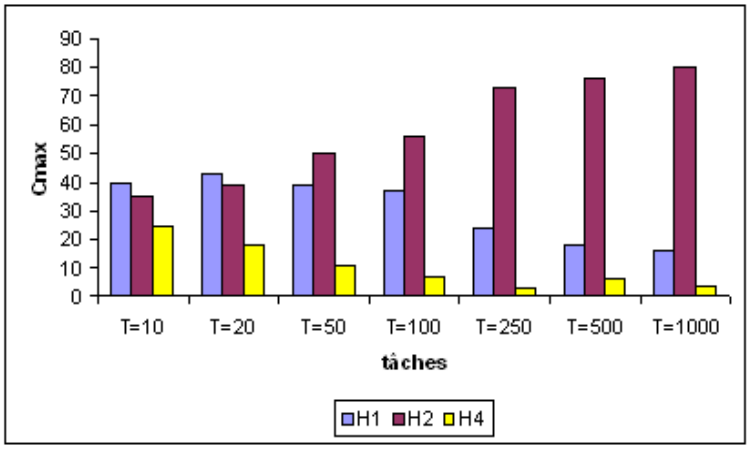

FIG. 4.3 – **Représentation des résultats de la loi uniforme par histogramme**

Nous remarquons que l'heuristique $H1$ est la mieux classée pour $T = 10$ et $T = 20$, l'heuristique $H2$ est en deuxième position suivie de l'heuristique $H4$. A partir de $T = 50$, $H2$ donne la meilleure valeur de C_{max} par rapport à $H1$ et l'heuristique $H4$ est en troisième position. Nous remarquons également qu'il est rare d'obtenir des solutions optimales.

- $p_i \in [1, 50]$ et $nb_i \in [1, 10]$:

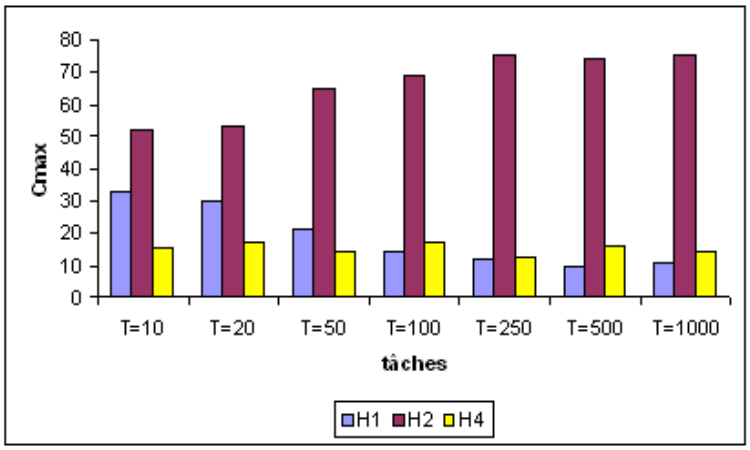

FIG. 4.4 – **Représentation des résultats de la loi uniforme par histogramme**

En augmentant le nombre d'opérations de tâches $nb_i \in [1, 10]$, l'heuristique $H2$ reste toujours la meilleure calssée, puis $H1$, ensuite $H4$ jusqu'à ce qu'à $T = 50$. A partir de $T = 100$, l'heuristique $H4$ prend la deuxième position suivie de $H1$. Pour $nb_i \in [1, 10]$, les heuristiques ne fournissent plus des solutions optimales à partir de $T = 50$.

En ce qui concerne la durée moyenne d'exécution de chaque heuristique, elle varie selon le nombre de tâches. En augmentant le nombre de tâches (opérations) et la durée de traitement, la durée augmente et varie de quelques millisecondes à quelques secondes au maximum, mais $H4$ possède toujours la durée d'exécution la plus élevée suivie de $H1$ et de $H2$.

2. La loi normale

- $p_i \in [1, 30]$ et $nb_i \in [1, 5]$:

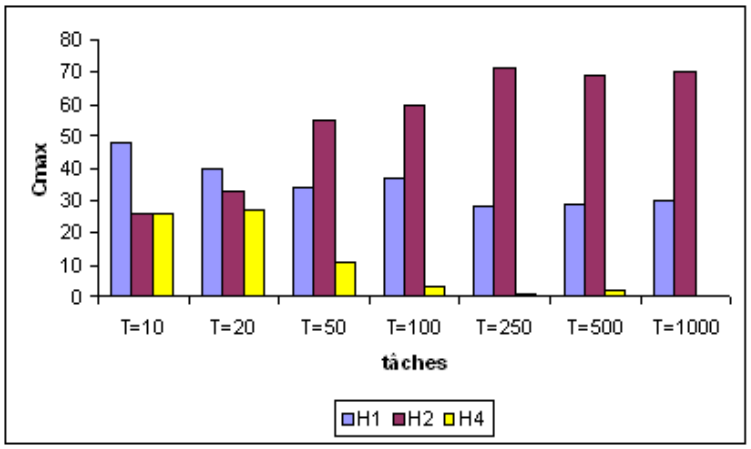

FIG. 4.5 – **Représentation des résultats de la loi normale par histogramme**

l'heuristique $H1$ donne la meilleure valeur de C_{max} ensuite $H2$ puis $H4$ pour un petit nombre de tâches ($T = 10$ et $T = 20$). Dès qu'on augmente le nombre de tâches (à partir de $T = 50$), $H2$ est la meilleure classée et $H4$ est en troisième position et son pourcentage devient nul pour $T = 1000$.

- $p_i \in [1, 50]$ et $nb_i \in [1, 5]$:

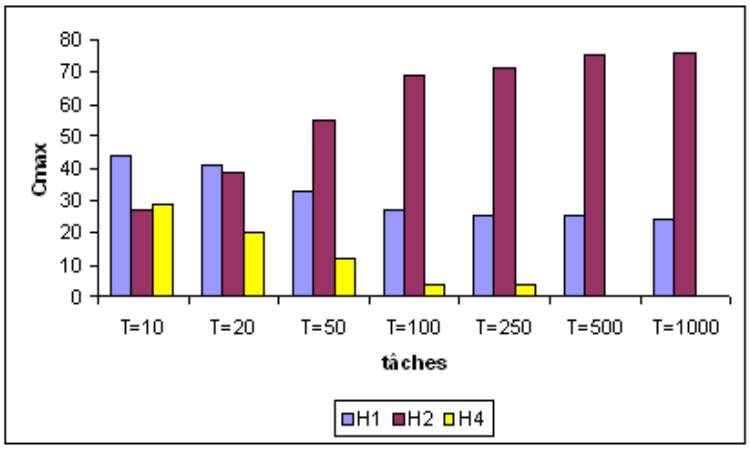

FIG. 4.6 – **Représentation des résultats de la loi normale par histogramme**

En augmentant la durée de traitement des opérations $p_i \in [1, 50]$, le même raisonnement est déduit sauf que le pourcentage devient nul à partir de $T = 500$.

- $p_i \in [1, 30], nb_i \in [1, 10]$ et $p_i \in [1, 50], nb_i \in [1, 10]$:

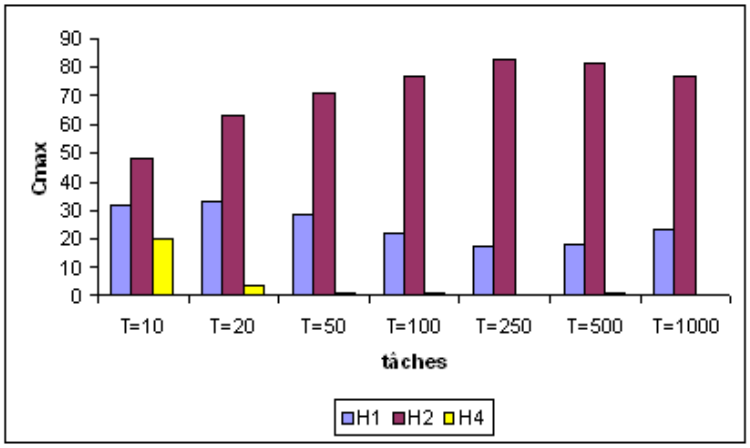

FIG. 4.7 – **Représentation des résultats de la loi normale par histogramme**

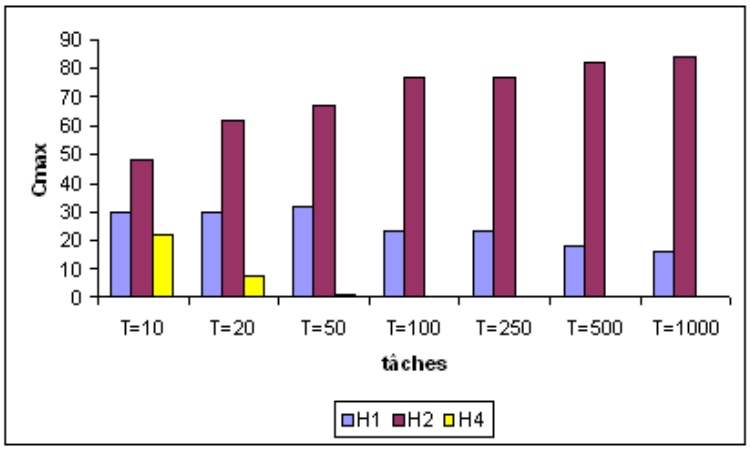

FIG. 4.8 – **Représentation des résultats de la loi normale par histogramme**

En augmentant le nombre d'opérations $nb_i \in [1, 10]$ l'heuristique $H2$ donne la meilleure valeur de C_{max} pour un petit et un grand nombre de tâches. L'heuristique $H4$ ne donne pratiquement pas de meilleure valeur de C_{max} à partir de $T = 100$. Des cas d'optimalités sont obtenus dans certains cas.

En augmentant le nombre d'opérations de tâches, la durée moyenne d'exécution augmente pour atteindre quelques secondes et millisecondes pour $nb_i \in [1, 10]$.

3. La loi exponentielle

- $p_i \in [1, 30]$ et $nb_i \in [1, 5]$:

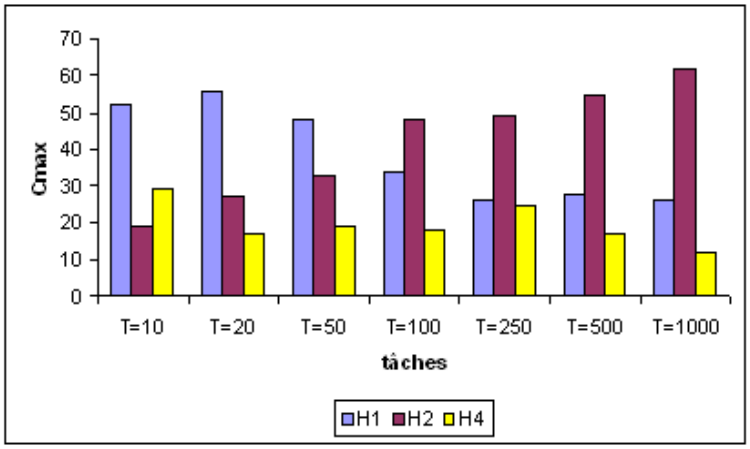

FIG. 4.9 – **Représentation des résultats de la loi exponentielle par histogramme**

Nous remarquons que l'heuristique $H1$ fournit un meilleur pourcentage de C_{max} pour $T = 10$, $T = 20$ et $T = 50$. L'heuristique $H4$ est meilleure que $H2$ pour $T = 10$, ce qui est le contraire dans le cas où $T = 20$, $T = 50$. A partir de $T = 100$, $H2$ devient la mieux classée ensuite $H1$ et en dernier $H4$.

- $p_i \in [1, 50]$ et $nb_i \in [1, 5]$:

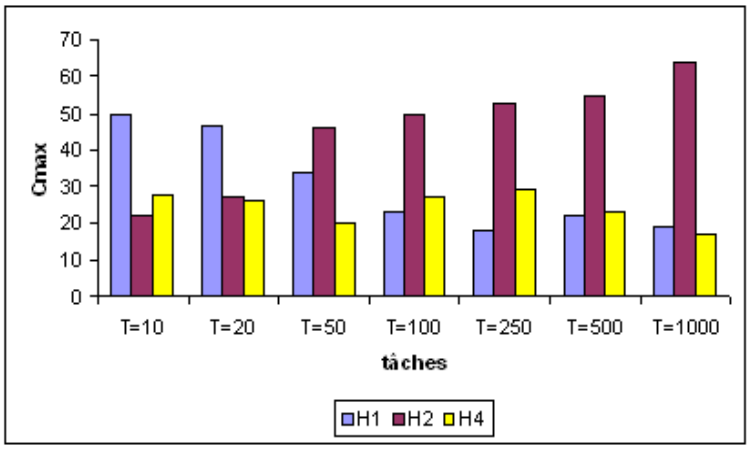

FIG. 4.10 – **Représentation des résultats de la loi exponentielle par histogramme**

Si on fait varier les durées de traitement des opérations ($p_i \in [1, 50]$), l'heuristique $H1$ fournit un meilleur pourcentage de C_{max} pour $T = 10$, $T = 20$. A partir de $T = 50$, $H2$ devient la mieux classée ensuite $H4$ (pour $T = 100$, $T = 250$ et $T = 500$) et en dernier $H1$. Mais pour $T = 1000$ l'heuristique $H1$ est la meilleure classée que $H4$.

- $p_i \in [1, 30]$ et $nb_i \in [1, 10]$:

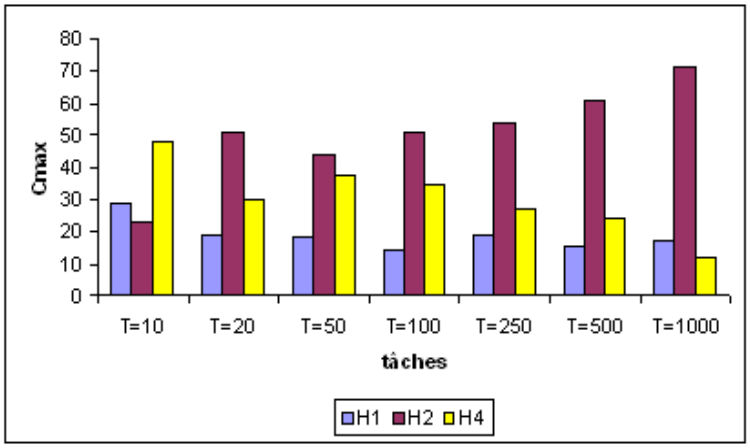

FIG. 4.11 – **Représentation des résultats de la loi exponentielle par histogramme**

Nous constatons que pour $T = 10$, l'heuristique $H4$ donne la meilleure valeur de C_{max} mais en augmentant le nombre de tâches (à partir de $T = 20$), $H2$ est en meilleure position. L'heuristique $H4$ est mieux classée que $H1$ jusqu'à ce que le nombre de tâches soit égale à 1000 où $H1$ devient meilleure que $H4$.

- $p_i \in [1, 50]$ et $nb_i \in [1, 10]$,

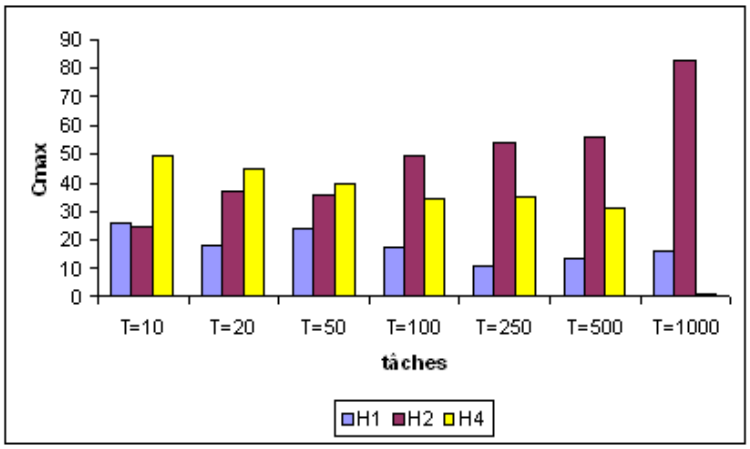

FIG. 4.12 – **Représentation des résultats de la loi exponentielle par histogramme**

Dans le cas nous remarquons que l'heuristique $H4$ fournit la meilleure valeur de C_{max} suivie de $H2$ et $H1$, pour $T = 10$, $T = 20$ et $T = 50$. A partir de $T = 100$ la meilleure valeur de C_{max} est obtenue par l'heuristique $H2$, puis l'heuristique $H4$ et en dernière position c'est l'heuristique $H1$. L'heuristique $H1$ est meilleure que $H4$ pour un nombre de tâches $T = 1000$.

Les solutions optimales sont rarement obtenues par les heuristiques proposées . La durée moyenne d'exécution est importante en augmentant le nombre d'opérations. L'heuristique

$H4$ possède toujours la durée moyenne d'exécution la plus élevée et ne dépassant pas quelques secondes.

4. La loi de poisson

- $p_i \in [1, 30]$ et $nb_i \in [1, 5]$:

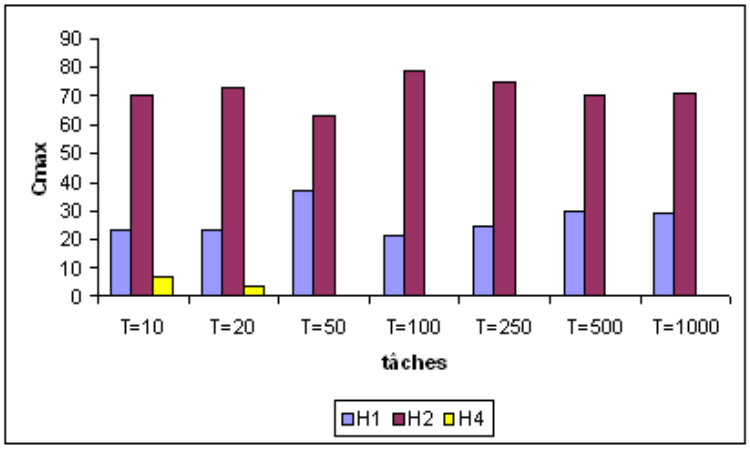

FIG. 4.13 – **Représentation des résultats de la loi de poisson par histogramme**

On constate que l'heuristique $H2$ fournit de meilleure valeurs de C_{max} allant de $T = 10$ jusqu'à $T = 1000$. L'heuristique $H1$ est mieux classée par rapport à $H4$. A partir de $T = 50$, l'heuristique $H4$ fournit des valeurs nuls de C_{max}.

- $p_i \in [1, 30]$ et $nb_i \in [1, 10]$:

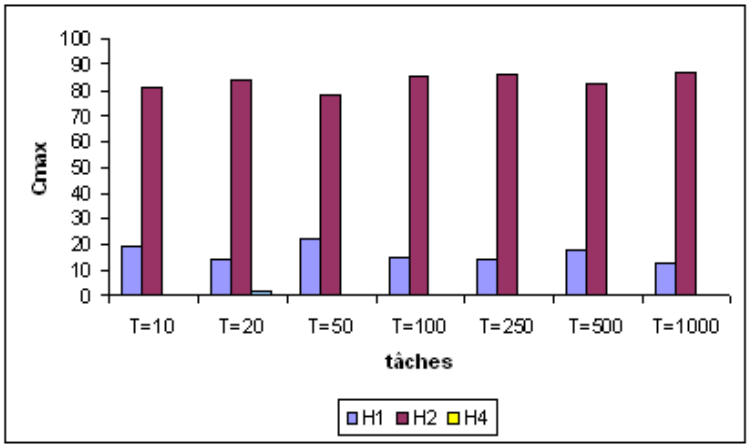

FIG. 4.14 – **Représentation des résultats de la loi de poisson par histogramme**

Le même raisonnement est déduit en augmentant le nombre d'opérations ($nb_i \in [1, 10]$), sauf que les valeurs de C_{max} s'annulent pratiquement pour toutes les valeurs de nombre de tâches à l'exception de $T = 20$.

- $p_i \in [1, 50]$ et $nb_i \in [1, 5]$:

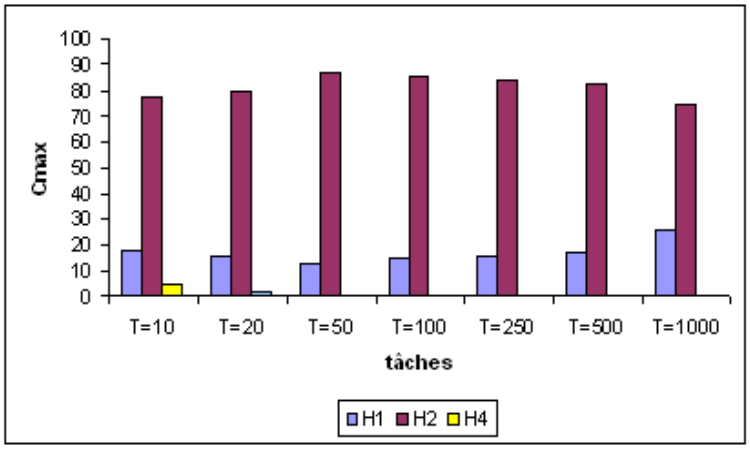

FIG. 4.15 – **Représentation des résultats de la loi de poisson par histogramme**

Un raisonnement similaire à celui du premier cas est déduit pour ce cas.

- $p_i \in [1, 50]$ et $nb_i \in [1, 10]$:

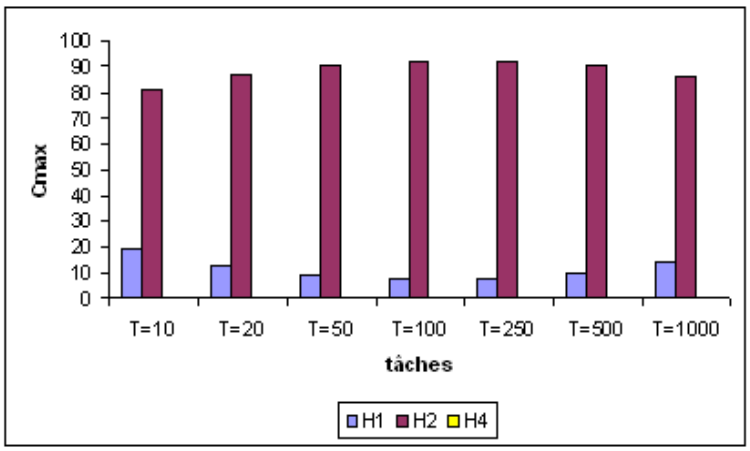

FIG. 4.16 – **Représentation des résultats de la loi de poisson par histogramme**

Un autre raisonnement est également similaire à celui du second cas est déduit pour ce cas.

On déduit que si on augmente le nombre d'opérations $nb_i \in [1, 10]$, l'heuristique $H4$ ne fournit pas de meilleure valeur de C_{max} à partir de $T = 20$, quelques soient les valeurs de $p_i \in [1, 30], p_i \in [1, 50]$ et $H2$ reste la meilleure.

Les solutions optimales sont pratiquement rares obtenues. Les durées moyennes de traitement croient en augmentant le nombre d'opérations et $H4$ est l'heuristique qui possède la durée d'exécution la plus élevée.

4.4 Conclusion

Après la réalisation de plusieurs tests, nous avons constaté que les résultats diffèrent d'une loi à une autre. Nous avons remarqué que l'heuristique $H1$ s'adapte mieux dans le cas d'un petit nombre de tâches (10, 20 et 50 tâches) et au petit nombre d'opérations ($nb_i \in [1, 5]$) pour les lois uniforme, normale et exponentielle. En augmentant le nombre d'opérations des tâches, l'heuristique $H4$ est la mieux appropriée dans le cas de la loi exponentielle et $H2$ dans les cas des lois uniforme et normale. Pour un grand nombre de tâches, l'heuristique $H2$ est la mieux positionnée. En ce qui concerne la loi de poisson, l'heuristique $H2$ est la mieux appropriée pour un petit et grand nombre de tâches et d'opérations.

Conclusion générale

La recherche en ordonnancement a beaucoup approffondi ses résultats ces dernières années. Les contraintes prises en compte dans les travaux récents sont de plus en plus complexes.

Dans ce mémoire, nous avons présenté deux problèmes d'ordonnancement de tâches non préemptifs, de type flow shop dont le but est la minimisation de la date de fin de traitement des tâches. Le premier problème est l'ordonnancement sur deux machines avec recirculation de tâches sur la seconde machine. Nous avons montré que ce problème est polynomial et nous avons donné un algorithme pour sa résolution. Le second problème consiste en ordonnancement sur deux étages, avec une machine sur le premier étage et deux machines identiques sur le second. Les tâches peuvent être traitées un nombre fini de fois sur le deuxième étage.

Le second problème traité est tiré d'une application réelle. Une série de pièces, avant être traitées sur un ensemble de machines paralèlles subissent un test ou un contrôl de qualité et de conformité. Les pièces non conformes ou de mauvaise qualité sont rejetées.

Nous avons modélisé ce problème sous forme d'un programme linéaire en variables réelles et bivalentes. Le nombre de variables et de contraintes de ce modèle est très important.

Nous avons montré que le problème sans recirculation, pour des durées de traitement des tâches égales à 1 sur le premier étage, est NP-difficille. Nous avons également montré que le problème avec recirculation, pour des durées de traitement des tâches égales à 1 sur le premier et le second étage, est polynomial et nous avons proposé un algorithme pour sa résolution.

Nous avons aussi proposé quatre heuristiques pour la résolution du second problème défini et chaqu'une des heuristiques diffère de l'autre dans la règle utilisée pour déterminer la séquence des tâches sur le premier étage.

Des expérimentations numériques ont été effectuées en réalisant une application informatique implémentée sous délphi7. Ces expérimentations se résument dans l'application des trois heuristiques ($H1$, $H2$ et $H4$) sur plusieurs instances dont les durées de traitement des tâches sont générées aléatoirement suivant différentes lois de probabilité.

Comme perspectives à notre travail, d'autres méthodes de résolution peuvent prendre part, en faisant appel aux méthodes exactes (programmation dynamique, procédure par séparation et évaluation, etc) et aux métaheuristiques (recuit simulé, recherche tabou, algorithmes génétiques, etc) que nous envisageons d'utiliser pour comparer les résultats de ces méthodes avec ceux que nous avons déjà obtenus. Nous souhaitons également se rapprocher de la réalité par l'illustration sur un problème de production concret. C'est-à-dire considérer un vrai problème industriel d'une entreprise de production et lui appliquer les différentes méthodes que nous avons développé.

Bibliographie

[1] Aghezzaf E. H., Artiba A., Moursli O., and Tahon C., *Hybrid flowshop problems, a decomposition based heuristic approach*, In Proceedings of International Conference on Industrial Engineering and Production Management (IEPM'95), Marrakech, FUCAM/IFIP/INRIA, pp. 43-56, 1995.

[2] Baptiste P. et Hguny L. K., *A branch and bound algorithm for the* $F/no-idle/C_{max}$, Proceedings of The International Conference on Industrial Engineering and Production Management (IEPM'97), Vol.1, pp. 429-438, 1997.

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

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

[5] Benoît M. and Billaut J-C., *Characterization of the optimal solutions of the* $F2//Cmax$ *scheduling problem*, Dans Second Conference on Management and Control of Production and Logistic (MCPL 2000), Grenoble, France, 2000.

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

[7] Billaut J-C. et Sylvain B., *Problème d'ordonnancement multicritère dans un flow shop hybride avec recirculation*, 3^e Conférence Francophone de Modélisation et Simulation, MOSIM'01, 2001.

[8] Billaut J-C. et Sylvain B., *A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation*, European Journal and Operational Research, 159, 651-662, 2004.

[9] Blazewicz J., Dror M., Pawlak G. and Stecke K. E., *A note on flexible flow shop scheduling with two-stages*, Working paper, 9409-15, 1994.

[10] Blazewicz J., Ecker K., Pesch E., Schmidt G. and Weglargz J., *Scheduling Computer and manufacturing Process*, Springer Verlag, Berlin, Heidelberg, 1996.

[11] Brah S. A. and Hunsucker J. L., *Branch and bound algorithm for the flowshop with multiple processors*, European Journal of Operational Research, 51, p. 88-99, 1991.

[12] Campbell H. G., Dudek R. A. et Smith M. L., *A new heuristics for the n jobs m machines sequencing problem*, Management Science, Vol. 16B, pp. 630-637, 1970.

[13] Carlier J. et Chretiennne P., *Problèmes d'ordonnancement : modélisation, complexité et algorithmes*, Collection Etude et Recherche en Informatique, Masson Edition, 1988.

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

[15] Cook S., *The complexity of theorem-proving procedures*, In Third ACM Symposium on Theory of computing, pp. 151-158, 1971.

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

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

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

[19] Garey M. R., Johnson D. S. et Sethi R., *The complexity of the flow shop and the job shop scheduling*, Mathematics of Operational Research, Vol. 1, pp. 117-129, 1976.

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

[21] Goldberg D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, USA, 1989.

[22] Gourgand M., Grangeon N. and Norre S., *Metaheuristics for the deterministic hybrid flow shop problem*, RAIRO-APII-JESA, 34(9), p.1107-1135, 2000.

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

[24] Guinet A., Solomon M. M., Kedia P. K., and Dussauchoy A., *computational study of heuristics for two-stage flexible flow-shops*, International Journal of Production Research, 34(5), p. 1399-1416, 1996.

[25] Gupta J. N. D., *Two-stage, hybrid flow shop Scheduling problem*, Journal of the Operational Research Society, Vol. 39, p. 359-364, 1988.

[26] Gupta J. N. D., Hariri A. M. A. and Potts C. N., *Scheduling a two-stage hybrid flow shop with parallel machines at the first stage*, Annals of Operational Research, 69, 171-191, 1997.

[27] Gupta J. N. D. et Tunc E. A., *Schedules for a two-stage hybrid flowshop with parallel machines at the second stage*, International Journal of Production Research, 29, n7, p. 1489-1502, 1991.

[28] Haouari M. and M'Hallah R., *Heuristic algorithms for the two-stage hybrid flowshop problem*, Operation Research Letters, 21(1), p. 43-53, 1997.

[29] Hejazi S. R. and Saghafian S., *Flow shop-Scheduling problems with makespan criterion : a review*, International Journal of Production Research, Vol. 43, No. 14, pp. 2895-2929, 2005.

[30] Johnson M. S., *Optimal two and three stage production schedules with setup times included*, Naval Research logistics Quartely, 1, 61-67, 1954.

[31] Karp. R, *Complexity of computer Computations,* R. E. Miller, J.W. Thatcher(eds.), Plenium Press, New York, 1972.

[32] Langston M. A., *Interstage transportation planning in the deterministic flow-shop environment*, Operations Research, 35, n4, p. 556-564, 1987.

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

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

[35] Moursli O. and Pochet Y., *A branch and bound algorithm for hybrid flow shop*, International Journal Production Economics, 64, 113-125, 2000.

[36] Negenman E. G., *Local search algorithms for the multiprocessor flow shop scheduling problem*, European Journal of Operational Research, 128, p. 147-158, 2001.

[37] Nowicki E. and Smutnicki C., *The flow shop with parallel machines : a tabu search approach*, European Journal of Operational Research, 106, 226-253, 1998.

[38] Page S., *An approach to the scheduling of jobs on machines*, J. Roy. Statist. Soc., 23, p. 484-492, 1961.

[39] Pinedo M., *Scheduling : Theory, Algorithms and Systems*, Prentice Hall Series, New Jersey, 1995.

[40] Pranzo M., *Batch Scheduling in a Two-machine flow shop with limitted buffer and sequence dependent setup times and removal times*, European Journal of the Operational Research, 153, 581-592, 2004.

[41] Rémy D., *Amélioration des performances des systèmes de production : apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles*, habilitation à diriger des recherches, Décembre 2004.

[42] Riane F., *Scheduling hybrid flow shop : Algorithms and Applications*, PhD thesis, Facultés Universitaires Catholiques de Mons, 1998.

[43] Sadfi C., *Problèmes d'ordonnancements avec minimisation des encours*, thèse de doctorat, spécialité génie industriel, Institut national polytechnique de Grenoble, février 2002.

[44] Sakarovitch M., *Optimisation Combinatoire : Programmation discrète*, Herman, Paris, 1984.

[45] Taillard E., *Some Efficient Heuristic Methods for the flow shop Sequencing Problem*, European Journal of the operational Research, Vol. 47, pp. 65-74, 1990.

[46] Tanarv V. S., Sotskov Y. N. et Strvsevitch V. A., *Scheduling Theory Multistage System*, Kluwer Academic Publishers, ISBN O-7923-2854-X, 1994.

[47] Vignier A., Billaut J-C. et Proust C., *Les problèmes d'ordonnancement de type flow shop hybride : État de l'art*, Rairo recherche opérationnelle, Vol. 33, n°2, pp. 117-183, 1999.

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