N° d'ordre : 09/2010-M/MT

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENNE

FACULTÉ DE MATHEMATIQUES

Mémoire présenté pour l'obtention du diplôme de Magister en Mathématiques

Spécialité : Recherche Opérationnelle

Par

AMROUCHE Karim

THÈME

Flow shop à deux machines avec recirculation

Soutenu publiquement, le 24/10/2010, devant le jury composé de :

Mlle	BOUCHEMAKH Isma,	Professeur,	à L'U.S.T.H.B.,	Présidente
Mr	BOUDHAR Mourad,	Professeur,	à L'U.S.T.H.B.,	Directeur de mémoire
Mr	BOUROUBI Sadek,	Professeur,	à L'U.S.T.H.B.,	Examinateur
Mr	OULAMARA Ammar,	MC, HDR,	à L'E.M. Nancy,	Examinateur
Mlle	MEZIANI Nadjat,	MAB,	à L'U.A.M. Béjaia,	Invitée

Table des matières

Remerciements

Ce Mémoire, intitulé "Flow shop à deux machines avec récirculation", s'est déroulé dans le cadre de l'Ecole Doctorale en Recherche Opérationnelle de la Faculté de Mathématique de l'**USTHB** durant l'année universitaire 2009/2010 sous la direction du Professeur BOUDHAR Mourad.

Mes premiers remerciements iront au Directeur du mémoire Monsieur BOUDHAR Mourad pour m'avoir soutenu le long de cette année de recherche, ainsi que pour sa précieuse aide, sa disponibilité et ses conseils éclairés pour la réalisation de ce travail. J'aimerais lui adresser mes plus vifs remerciements.

J'adresse mes remerciements au Professeur BOUCHEMAKH Isma de l'USTHB pour avoir bien voulu me faire l'honneur de présider le jury de ce mémoire.

Je remercie le Professeur BOUROUBI Sadek de l'USTHB, Monsieur OULAMARA Ammar, Maitre de conférences HDR de l'Ecole des Mines de Nancy, et Mademoiselle MEZIANI Nadjat, Maître Assistante à l'université de Béjaia, pour avoir accepté d'être examinateurs de mon travail.

Je ne pourrais clôturer ces remerciements sans me retourner vers les êtres qui mes sont le plus chers, qui ont eu un rôle essentiel et continu pendant ma réussite, et qui sans eux aucune réussite n'aurait été possible. J'adresse de tout mon cœur mes remerciements à ma famille, particulièrement à mes chers parents, mon frère et ma sœur, qui furent toujours présent et ont rempli ma vie, mon seul exemple, je leurs suis infiniment reconnaissant pour leurs amours et leurs soutiens.

Enfin, je ne pourrai oublier mes collègues de l'Ecole Doctorale qui m'ont apporté un grand réconfort de travail dans la joie et la bonne humeur, ainsi que tous les professeurs qui nous ont dispensé des cours pendant notre première année de post-graduation.

Résumé

Nous présentons un problème d'ordonnancement d'atelier de type flow shop. L'atelier étudié est constitué de deux machines avec duplication des tâches sur la première machine (recirculation sur la première machine). L'objectif est de construire un ordonnancement minimisant la date de fin de traitement de l'ensemble des tâches. Le problème général étant NP-difficile, nous montrons que des cas particuliers de ce problème sont polynomiaux et nous donnons des algorithmes optimaux pour les résoudre. Nous présentons également des heuristiques et une métaheuristique pour la résolution du problème général avec des expérimentations numériques.

Mots clés : Ordonnancement, makespan, flow shop, recirculation, heuristique, métaheuristique.

Abstract

We present a scheduling problem of type flow shop. The studied shop is constituted of two machines with job re-entrance at the first machine. The objective is to determine a feasible schedule with minimal makespan. The general problem being NP-hard, we prove that some special cases of this problem are polynomially solvable and we give optimal algorithms to solve them. We also present heuristics and a metaheuristic for the resolution of the general problem with numerical experimentations.

Keywords : Scheduling, makespan, flow shop, re-entrance,heuristics, metaheuristic.

Introduction générale

Du fait de la concurrence, les industries, pour survivre, se doivent d'être toujours plus compétitives. Cette compétitivité passe par la réduction des coûts et un gain de productivité. Les industries se sont donc tournées vers la recherche et le développement dans le but d'améliorer leurs processus et notamment ceux de fabrication. Ainsi, la branche de l'optimisation combinatoire, qu'est la théorie de l'ordonnancement, de par son large champ d'investigation, permet de répondre à certaines des problématiques industrielles. En effet, cette théorie concerne les problèmes d'allocation, dans le temps, d'un ensemble limité de ressources par un ensemble de tâches. La richesse d'interprétation des termes "ressource" et "tâche" permet l'application pratique à une multitude de cas. L'évolution et la mondialisation de la production ainsi que l'ouverture des marchés internationaux ont poussé les industriels à se diriger vers des systèmes de fabrication flexibles ; ceci conduit à mettre en place une logique industrielle qui remet en cause certains fondements des habitudes de la production. Ainsi, la gestion des ateliers de production a pris une certaine valeur ces dernières années afin d'améliorer la productivité et d'augmenter la réactivité. Ces nouveaux domaines d'application rendent plus complexes les problèmes d'ordonnancement des tâches ; ce qui exige la résolution rigoureuse de ces problèmes. En effet, par une meilleure utilisation et planification des ressources, un atelier de production peut réaliser une grande variété de produits et les couts peuvent être réduits ce qui constitue un objectif crucial. En particulier, la productivité peut être affectée par l'ordonnancement des opérations sur les machines. Le champ d'application de l'ordonnancement est large : la gestion de la production dans l'industrie, la gestion de projets, l'organisation des emplois du temps, la gestion de la charge des processeurs en informatique, etc. En raison de leur nature fortement combinatoire, l'étude des problèmes d'ordonnancement reste également d'un intérêt théorique toujours renouvelé, car il n'existe pas de méthode de résolution à la fois générale et de faible complexité algorithmique .

Gérer un système de production, c'est parvenir à décider, dans le détail, des tâches à

accomplir, des ressources à utiliser pour accomplir ces tâches, et des instants où ces tâches vont débuter : c'est ce que l'on appelle l'ordonnancement. Cette activité, destinée à permettre de répondre, au plus bas prix, aux demandes des clients, fait intervenir un très grand nombre de facteurs. Elle ne peut donc pas se dérouler sur une très longue période. Typiquement, un ordonnancement dépasse rarement, dans la pratique, la semaine. Cependant on comprend qu'il soit nécessaire de tenir compte des demandes qui se manifestent de l'horizon de la semaine. Si la demande d'une période est faible, alors que la demande de la période suivante est importante et dépasse la capacité du système, il est nécessaire de le savoir afin de produire à l'avance une partie de ce qui sera demandé plus tard. Résoudre un problème d'ordonnancement consiste à organiser ces tâches, c'est-à-dire à déterminer leurs dates de démarrage, et à leur attribuer des ressources, de telle sorte que les contraintes soient respectées, afin d'optimiser un certain objectif. En effet, la résolution d'un problème d'ordonnancement passe par deux étapes principales. La première étape consiste à identifier et à modéliser le problème en décrivant les contraintes qui doivent être respectées et mettant en valeur les critères à optimiser. La deuxième étape se traduit par la recherche de la méthode adéquate pour résoudre le problème considéré. Les problèmes d'ordonnancement, très variés, sont caractérisés par un grand nombre de paramètres relatifs aux tâches (préemptives ou non, indépendantes ou non), aux ressources (renouvelables ou consommables), aux types de contraintes portant sur les tâches (précédences, disjonctions), au(x) critère(s) d'optimalité (minimisation de la durée de l'ordonnancement, minimisation du retard maximal des tâches, etc.).

Dans ce travail nous considérons le problème d'ordonnancement de type Flow shop dont le but est de minimiser le makespan. Ce problème est un flow shop a deux machines avec recirculation des tâches sur la première machine. Le problème général étant NP-difficile, nous montrons que des cas particuliers de ce problème sont polynomiaux et nous donnons des algorithmes optimaux pour les résoudre. Nous présentons également des heuristiques et une métaheuristique pour la résolution du problème général avec des expérimentations numériques.

Ce mémoire est organisé en quatre chapitres. Le premier est consacré aux notions générales sur l'ordonnancement. Le second chapitre est réservé à la notation utilisée et à la position du problème. Le troisième chapitre est consacré à la modélisation mathématique et aux méthodes de résolutions. Le problème en général est NP-difficile, nous donnons des cas particuliers où le problème est polynomial. On propose également cinq heuristiques dont deux sont présentées dont l'article de Caixia Jing et al [6] que nous avons adapté

à notre problème. Dans ce même chapitre nous avons développé une métaheuristique. Toutes ces méthodes proposées ont été programmées sous " Microsoft visual C# " pour les besoins de l'expérimentation numérique. Dans le quatrième chapitre, on présente les résultats de ces testes sur des instances de différentes tailles (allant de 10 à 500 tâches) où les durées opératoires des tâches sont générées suivant une loi uniforme. On termine par une conclusion.

Chapitre 1

Généralités en ordonnancement

1.1 Introduction

Ordonnancer c'est programmer l'exécution d'un ensemble de tâches (travaux, tasks ou jobs en anglais) en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. Les problèmes d'ordonnancement apparaissent dans le suivi des projets, les ateliers de production, les emplois du temps, en informatique, etc. En effet, les tâches peuvent représenter des pièces mais également des projets, des programmes ou encore des activités, etc. De méme, les ressources peuvent représenter des machines industrielles mais également des processeurs ou des personnes, etc. Les différentes données d'un problème d'ordonnancement sont, donc, les tâches, les contraintes, les ressources et les objectifs. Une ressource est un moyen matériel, financier ou humain à disposition pour la réalisation d'une tàche.

Les contraintes représentent les limites imposées par l'environnement, tandis que l'objectif est le critére d'optimisation. Si tous les paramêtres des tâches sont connus à priori, le problème est dit déterministe.

Dans les problèmes de l'ordonnancement classique, deux hypothéses importantes sont communément considérées :

(a) à chaque instant, une machine ne peut traiter qu'une seule tàche à la fois et

(b) à chaque instant, une tàche ne peut être traitée que par, au plus, une machine.

1.2 Concepts de base en ordonnancement

Les tâches

Une tâche est une entité 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 $C_i = s_i + p_i$).

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 (supposé constante).

On distingue plusieurs types de ressources :

- Ressources renouvelables

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

- Ressources consommables

Une ressource est dite consommable si elle n'est plus disponible en méme quantité après avoir été utilisée par une ou plusieurs tâches (matiéres premières, budget, ...). La consommation globale (ou cumul) au cours du temps est limitées.

- 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).

1.2.1 Définitions, Description des tâches et Critéres d'optimalité

L'ensemble des n tâches est noté $T = \{T_1, T_2, ..., T_n\}$. L'ensemble des m machines est noté $M = \{M_1, M_2, ..., M_m\}$.

Caractérisation des machines

Il existe en général deux types de machines :

Les machines paralélles (parallel machines) : ces machines exécutent les mémes traitements et sont donc capables de traiter n'importe quelle tâche. En fonction de la vitesse de traitement, nous distinguons trois types de machines paralléles :

– **Les machines identiques (identical machines)** : pour ces machines, les vitesses de traitement des tâches sont toutes égales.

– **Les machines uniformes (uniform machines) :** pour ces machines, les vitesses de traitement des tâches sont différentes mais que la vitesse de chaque machine est constante et ne dépend pas des tâches de T.

– **Les machines générales (unrelated machines) :** pour ces machines, la vitesse de traitement d'une machine dépend de la tâche qui est traitée.

Les machines spécialisées ou dédiées (dedicated machines) : ces machines sont spécialisées à l'exécution de certaines tâches. Dans le cas de machines spécialisées, nous distinguons trois systémes ou modes de traitement :

– **Flow shop :** dans un systéme flow shop, les tâches de T doivent être traitées par toutes les machines dans un méme ordre. Le méme ordre de traitement pour toutes les tâches.

– **Open shop :** dans un systéme open shop, les tâches de T doivent ètre traitées par toutes les machines mais l'ordre du traitement n'est pas nécessairement le méme pour toutes les tâches, l'ordre de traitement est quelconque.

– **Job shop :** dans un systéme job shop, le sous-ensemble de machines devant traiter une tâche et l'ordre du traitement sont arbitraires, mais doivent être spécifiés à l'avance.

Description des tâches

Dans le cas de machines paralléles, chaque tâche peut être traitée par n'importe quelle machine. Si les machines sont spécialisées, chaque tâche $T_i \in T$ est divisée en un nombre fini d'opérations $O_{i1}, O_{i2}, ..., O_{iki}$. chacune pouvant nécessiter une machine différente (k_i étant le nombre d'opérations de la tâche T_i) :

– dans un systéme flow shop, le nombre d'opérations par tâche est égal à m ($k_i = m$ pour $i = 1, ..., n$) et leur ordre de traitement est tel que l'opération O_{ij} est traitée sur la

machine M_j avant l'opération O_{ij+1} traitée sur la machine M_{j+1}.

Chaque tâche $T_i \in T$ peut être caractérisée par les données suivantes :

Un vecteur temps de traitement (processing time) $p_i = \{p_{i1}, ..., p_{im}\}$: p_{ij} est le temps de traitement nécessaire à la machine M_j pour terminer le traitement de la tâche T_i (durée de traitement).

– Dans le cas de machines identiques, nous avons : $p_{ij} = p_i$ pour $j = 1, ..., m$.

– Dans le cas de machines uniformes, nous avons : $p_{ij} = p_i/b_j$ pour $j = 1, ..., m$ où b_j est le facteur vitesse de traitement de la machine M_j et p_i un temps de traitement standard.

Une date d'arrivée, de disponibilité ou au plus tôt (release date, ready time ou arrivaltime) r_i : c'est la date à laquelle la tâche T_i est préte à être traitée (date de lancement) si pour toutes les tâches de T, les dates d'arrivée sont toutes égales, on pose $r_i = 0$ pour $i = 1, ..., n$.

Une date échue ou au plus tard (due date) d_i : si le traitement de la tâche T_i doit être achevé avant la date d_i.

Une date limite ou absolue (deadline) $\tilde{d}_i$: si le traitement de la tâche T_i doit obligatoirement être achevé avant la date $\tilde{d}_i$.

Un poids (weight) w_i : qui exprime la priorité relative de la tâche T_i.

Critéres d'optimalité

Les paramêtres suivants peuvent être calculés pour chaque tâche $T_i, i = 1, ..., n$; traitée dans un ordonnancement (admissible ou réalisable) donné :

– **Une date d'achévement (Completion time)** c_i : c'est la date de fin de traitement de la tâche T_i.

– **Une durée de flot (Flow time)** f_i : c'est la somme du temps d'attente et du temps de traitement. C'est aussi la différence entre la date d'achévement et la date d'arrivée. $f_i = c_i - r_i$.

– **Une tardiveté, décalage temporel ou retard algébrique (lateness)** $\mathbf{l_i}$: c'est la différence entre la date d'achévement et la date échue. $l_i = c_i - d_i$.

– **Un retard (tardiness)** t_i : $t_i = max\{l_i, 0\}$.

– **Un indicateur de retard** u_i : $u_i = 1$ si $c_i > d_i$ 0 sinon.

1.2.2 Classification

Etant donné la diversité des problèmes de l'ordonnancement, un formalisme permettant de distinguer les différents problèmes de l'ordonnancement entre eux et de les classifier est utilisé. Ce formalisme comporte trois champs $\alpha/\beta/\gamma$ permettant de décrire les différentes entités d'un problème d'ordonnancement.

Champ α : ressources

Ce champ α décrit les machines utilisées : type de machines, nombre de machines et type d'atelier. Il est composé de deux sous-champs et désigné par $\alpha = \alpha_1\alpha_2$.

– $\alpha_1 \in \{\emptyset, P, Q, R, O, F, J\}$ décrit le type de machine.

_ $\alpha_1 = \emptyset$: Une seule machine.

– $\alpha_1 = P$: machines paralléles identiques.

– $\alpha_1 = Q$: machines paralléles uniformes.

– $\alpha_1 = R$: machines paralléles différentes.

– $\alpha_1 = O$: machines spécialisées (systéme open shop).

– $\alpha_1 = F$: machines spécialisées (systéme flow shop).

– $\alpha_1 = J$: machines spécialisées (syséme job shop).

– $\alpha_2 \in \{\emptyset, k\}$ dénote le nombre de machines.

– $\alpha_2 = \emptyset$: le nombre de machines est variable.

– $\alpha_2 = k$: le nombre de machines est égal à k (k est un entier positif non nul).

Champ β : contraintes

Ce champ β décrit les tâches et les contraintes liées à ces derniéres. Il est composé de 5 sous-champs et désigné par $\beta = \beta_1, \beta_2, \beta_3, \beta_4, \beta_5$.

$\beta_1 \in \{\emptyset, pmtn\}$: indique la possibilité de préemption des tâches.

– $\beta_1 = \emptyset$: la préemption des tâches n'est pas autorisée.

– $\beta_1 = pmtn$: la préemption des tâches est autorisée.

$\beta_2 \in \{\emptyset, prec, tree, chains, sp\}$: indique le type de contraintes de précédence.

– $\beta_2 = \emptyset$: pas de contraintes de précédence (les tâches sont indépendantes).

– $\beta_2 = prec$: les contraintes de précédence sont quelconques.

– $\beta_2 = tree$: les contraintes de précédence sont données sous forme d'un arbre.

– $\beta_2 = chains$: les contraintes de précédence sont données sous forme de chaînes.

– $\beta_2 = sp$: les contraintes de précédence sont données sous forme d'un graphe série-paralléle.

– d'autres types de graphes peuvent aussi être utilisés.

$\beta_3 \in \{\emptyset, ri\}$: décrit les dates de disponibilité.

– $\beta_3 = \emptyset$: toutes les dates de disponibilité sont nulles.

– $\beta_3 = r_i$: il existe des dates de disponibilité pour les tâches. Non identiques

$\beta_4 \in \{\emptyset, p_i = p, \underline{p} \leq p_i \leq \bar{p}\}$.

– $\beta_4 = \emptyset$: les temps de traitement des tâches sont quelconques et non identiques.

– $\beta_4 = p_i = p$: les temps de traitement des tâches sont tous égaux à p (constants).

– $\beta_4 = \underline{p} \leq p_i \leq \bar{p}$: le temps de traitement de chaque tâche est compris entre $\underline{p}$ et $\bar{p}$.

$\beta_5 \in \{\emptyset, d_i\}$.

– $\beta_5 = \emptyset$: il n'y a pas des dates échues pour les tâches ou il existe des dates échues pour les tâches dans le cas ou le critére d'optimisation est en fonction de ces derniéres.

– $\beta_5 = d_i$: il existe des dates échues pour les tâches.

Remarque La notation $\emptyset$ signifie qu'il ne faut mettre aucun symbole.

Champ γ : critére d'optimalité

Le troisième champ γ décrit le critére d'optimalité. On cherche à minimiser γ avec :

$\gamma \in \{C_{max}, \quad \bar{C}, Cw, T_{max}, T, T_w, L_{max}, U, U_w, ...\}$ tel que

1. $C_{max} = \max_{1 \leq i \leq n} c_i$ la longueur de l'ordonnancement (schedule length ou makespan).
2. $F = \frac{1}{n} \sum_{i=1}^{n} f_i$ le temps de flot moyen (mean flow time).
3. $L_{max} = max_{\ 1 \leq i \leq n}\{l_i\}$ la grande tardiveté (maximum lateness).
4. $N_T = \sum_{i=1}^{n} u_i$ le nombre de tâches en retard (number of tardy jobs).

5. $\bar{C} = \sum_{i=1}^{n} \frac{1}{n} C_i$ le temps de fin de traitement moyen (mean completion time).

6. $F_w = \frac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$ le temps de flot moyen pondéré (mean weighted flow time).

7. $Cw = \frac{\sum_{i=1}^{n} W_i C_i}{\sum_{i=1}^{n} w_i}$ le temps de fin de traitement moyen pondéré (mean weighted completion time).

8. $F_{max} = max_{1 \leq i \leq n}\{f_i\}$ le grand temps de flot (maximum flow time).

9. $T_{max} = max_{1 \leq i \leq n}\{t_i\}$ le grand retard (maximum tardiness).

10. $L = \frac{1}{n} \sum_{i=1}^{n} l_i$ la tardiveté moyenne (mean lateness).

11. la tardiveté moyenne pondérée (mean weighted lateness).

12. le retard moyen (mean tardiness).

13. le retard moyen pondéré (mean weighted tardiness).

1.3 Complexité des problèmes d'ordonnancement

La complexité des problèmes d'ordonnancement est définie suivant la complexité des méthodes de résolution et celle des algorithmes utilisés. Certains problèmes d'ordonnancement de taille relativement importante peuvent avoir un niveau de complexité si important que leur résolution devient trés difficile. Deux catégories de complexité sont distinguées : algorithmique et problématique.

Complexité algorithmique

Elle exprime une fonction du nombre d'opérations élémentaires de calcul effiectuées par la méthode ou par l'algorithme de résolution en fonction du nombre des données du problème traitée [25]. La complexité d'un algorithme est définie par le temps de calcul nécessaire à son exécution pour la résolution d'un problème d'une taille donnée. Ce temps est, en réalité, exprimé en fonction du nombre d'instructions élémentaires nécessaires à l'exécution de l'algorithme par le processeur. Le choix de cette mesure est justifié par le fait

que le nombre d'instructions est indépendant de la puissance des machines qui détermine le temps de calcul effectif. La théorie de la complexité s'intéresse, entre autres, à l'ordre de grandeur de la complexité dans le pire des cas qui est dite en grand O. Pour estimer la complexité d'un algorithme, en grand O, il suffit de borner le nombre d'instructions élémentaires qu'il engendre. Cette borne est exprimée en fonction de la taille des données n, et lorsqu'elle s'écrit en $p(n)$ tel que $p(n)$ est une fonction polynomiale, l'algorithme est dit polynomial et sa complexité est donnée par $O(p(n))$. Notons que lorsque celle-ci est linéaire la complexité est également dite linéaire. Il existe d'autres types de complexité, par exemple : l'exponentielle soit en $O(a^n)$. Les algorithmes de complexité en $O(n!)$ ou en $O(n^{logn})$ sont également considérés comme des algorithmes de complexité exponentielle.

Complexité problématique

Cette notion est liée à la difficulté du problème à résoudre et au nombre d'opérations élémentaires qu'un algorithme déterministe doit effectuer pour trouver l'optimum en fonction de la taille du problème.

Classification des problèmes

Pour des raisons de simplicité et techniques aussi, la théorie de la complexité se limite juste à l'étude des problèmes de décision.

Définition 1.1 *Un problème de décision est un problème dont la solution est formulée en termes oui/non.*

De cette définition, ce qui importe c'est juste l'existence de la solution. Cette restriction aux problèmes de décision est justifiée par le fait que les autres problèmes qui ne sont pas de décision, comme les problèmes d'optimisation, peuvent être facilement transformés en un problème de décision équivalent.

Le but de la théorie de la complexité est la classification des problèmes de décision suivant leur degré de difficulté de résolution. Dans la littérature, il existe plusieurs classes de complexité, mais les plus connues sont les suivantes.

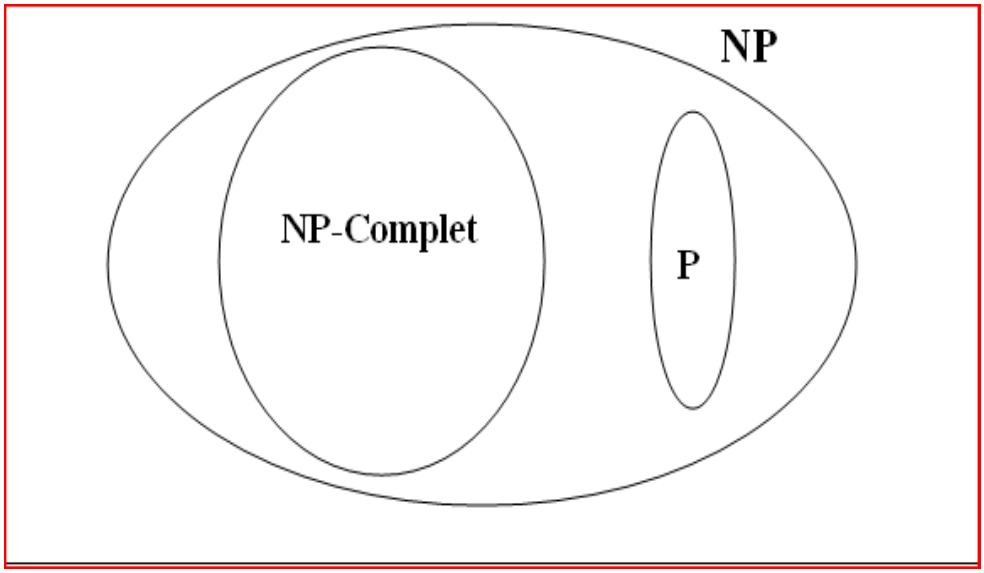
– **Les problèmes les plus difficiles :** Ce sont les problèmes pour lesquels on ne connait pas d'algorithme qui permet de décider si la réponse à une question posée est oui ou non. Ils sont également dits indécidables.

– **Les problèmes de la classe P :** un problème de décision est dit polynomial s'il existe

un algorithme de complexité polynomiale pour sa résolution. Mais, ne pas connaitre un tel algorithme ne signifie pas forcément qu'il n'existe pas.

– **Les problèmes de la classe NP :** La classe NP possède une définition moins naturelle que celle de P, et son nom est trompeur : il ne signifie pas « non polynomial », mais polynomial non-déterministe ou « Non deterministic Polynomial time » en anglais. Elle regroupe les problèmes qui peuvent être résolus en temps polynomial par des algorithmes non déterministes (un algorithme est dit non déterministe s'il comporte des instructions de choix). 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 exponentiel.

– **Les problèmes NP-complets :** Cette classe (NPC) concerne les problèmes les plus difficiles de la classe NP. Elle est basée sur la notion de réduction. En effet, un problème de décision est dit NP-complet si tout problème de la classe NP peut se réduire polynomialement à lui.



Les différentes classes

Quelques problèmes NP-complets – **problème du stable :** trouver, dans un graphe fini, un ensemble de m sommets non reliés (m étant donné).

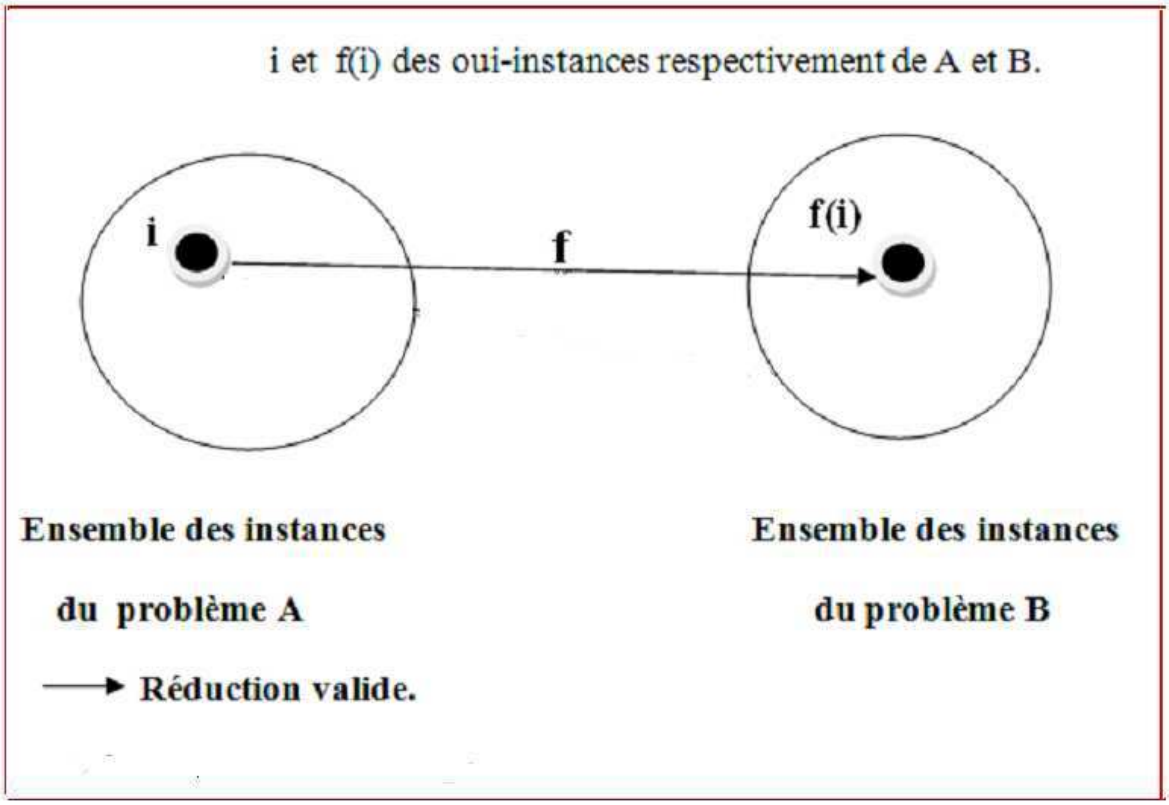
– **problème du sac à dos :** soient un ensemble S de nombre entiers positifs et un entier M. Existe-t-il une partie A de S telle que $\sum_{i \in A} a_i = M$?

– **problème du cycle hamiltonien :** soit un graphe fini. Existe-t-il un cycle de longueur n passant une et une seule fois par tous les n sommets ?

– **problème de 3-coloration :** soit un graphe fini. Peut-on colorier les sommets du graphe à l'aide de 3 couleurs de telle manière que deux sommets adjacents n'aient pas la méme couleur ?

– **problème des machine paralléles** : Soient m machines paralléles identiques et n tâches. Chaque tàche i doit être exécutée sans interruption par une des m machines pendant un temps p_i . Existe-t-il une affectation des n tâches sur les m machines de telle maniére que le temps de fin de toutes les tâches ne dépasse pas un temps T?

problèmes NP-difficiles Un problème d'optimisation est dit NP-difficile si le problème de décision associé P' est NP-complet.



La réduction polynomial

Concept de réduction

La réduction convertit un problème en un autre dans le but de pouvoir utiliser la solution du second problème pour résoudre le problème initial. La réduction sert à résoudre un problème à l'aide d'un second problème, mais permet aussi de montrer la difficulté de résolution d'un problème en le transformant en un autre dont la complexité est déjà connue. Une réduction est une transformation qui à une instance du problème A fait

correspondre une instance du problème B qui a la méme réponse. De plus, une oui-instance est une instance d'un problème de décision dont la réponse est oui. Dans le cas d'une réduction polynomiale, comme le montre la figure ci-dessus, la transformation, notée f, transforme chaque oui-instance i de A en une oui-instance de $f(i)$ de B. f étant polynomiale.

Comment prouver la NP-complétude d'un problème

L'étude de la complexité des problèmes d'ordonnancement conduit au choix de leurs méthodes de résolution. Il est donc essentiel de savoir si un problème est « facile » ou « difficile ». Si un problème semble $NP - complet$, il est nécessaire d'en établir la preuve. La liste suivante présente les étapes usuelles de l'établissement de cette preuve :

– transformer le problème d'optimisation en problème de décision P',

– montrer que P' est dans NP,

– choisir un problème P'' $NP - complet$, permettant de faire la réduction,

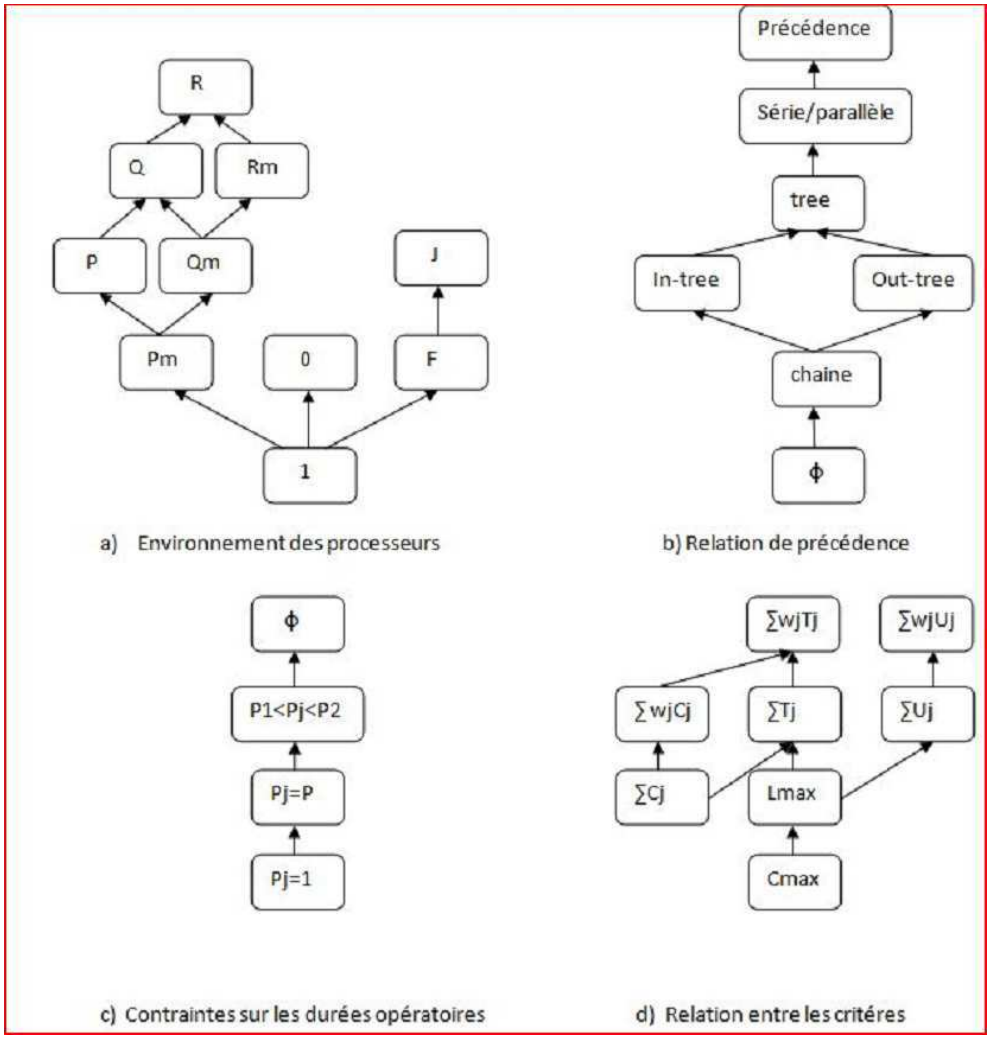
– construire une fonction f de P'' vers P' de telle sorte que I est une oui-instance de P'' si et seulement si $f(I)$ est une oui-instance de P',

– montrer que f est une réduction polynomiale.

Hiérarchie de complexité pour les problèmes d'ordonnancement

Suite à la hiérarchie de la complexité, et puisque les problèmes d'ordonnancement sont aussi des problèmes d'optimisation, la notion de réduction introduite plus tôt peut leur être appliquée. Une hiérarchisation des problèmes d'ordonnancement est donc possible. Plusieurs hiérarchies de complexité sont possibles suivant la ou les caractéristiques étudiées à savoir environnement des machines, relations de précédence, contraintes sur les durées opératoires et relations entre les critéres. Pour une représentation plus compléte de ces relations, il est possible de se référer à Pinedo [23] . A titre d'exemple , une hiérarchie utile à connaître est celle générale des problèmes ayant des configurations machines différentes dans des environnements identiques. Comme on peut le constater, la configuration la plus simple est celle qui est à une machine, puis l'ajout d'autres machines et de différentes contraintes augmentent la complexité. Nous exposons, dans la figure suivante certaines hiérarchies existantes entre différents problèmes. Le cas a) indique la hiérarchie de compléxité 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 paralléle ou en série. La difficulté augmente encore, si le nombre de machines est non borné et si l'on considére, non plus des machines identiques, mais des machines uniformes ou générales. En ce qui concerne les relations de précédence, le cas b) nous montre clairement les degrés de complexité. Le cas c) nous donne l'évolution de la difficulté, en fonction des valeurs des durées opératoires. Enfin, le cas d) donne la relation existant entre les principaux critéres d'optimisation.



Hiérarchie de complexité entre différents problèmes

1.4 Méthodes de résolution des problèmes d'ordonnancement

A coté des algorithmes classiques de l'optimisation combinatoire ont été développées un grand nombre d'heuristiques (algorithmes approchés polynomiaux), qui permettent d'obtenir un ordonnancement pour lequel la valeur du critére n'est pas trés éloignée de la valeur optimale. De nombreuses méthodes de résolution ont été développées en Recherche Opérationnelle (RO) et en Intelligence Artfficielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories :les méthodes exactes et les méthodes approchés.

Méthodes exactes

On peut définir une méthode exacte comme une méthode qui fournit une solution optimale pour un problème d'optimisation. L'utilisation de ces méthodes s'avére particuliérement intéressante dans le cas des problèmes de petite taille. Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de maniére implicite, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tot possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de Séparation et Evaluation Progressive (SEP) ou les algorithmes avec retour arriére. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrés réalisés (notamment en matiére de programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution, risquent d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante.

Procédure de Séparation et d'Evaluation (PSE)

Les procédures par séparation et évaluation sont également appelées « branch and bound », elles consistent à décomposer l'ensemble des solutions en sous-ensembles de solutions partielles. Pour un problème de minimisation, le calcul d'une borne inférieure pour chaque solution partielle permet d'évaluer sa qualité. Si cette borne inférieure est supérieure ou égale à la meilleure borne trouvée, on peut couper la branche correspondant à la solution

partielle car il n'existe pas de meilleure solution dans ce sous-ensemble. La difficulté de cette méthode réside dans le calcul d'un minorant suffisamment bon pour éliminer le plus rapidement possible un maximum de branches et atteindre un optimum. Pour les problèmes d'ordonnancement de grande taille, cette technique risque de générer une arborescence conséquente à explorer. Pour remédier à un tel problème, la borne utilisée doit être la plus fine possible.

Programmation dynamique

Elle se base sur le principe de Bellman : « Si C est un point qui appartient au chemin optimal entre A et B, alors la portion de ce méme chemin allant de A à C est le chemin optimal entre A et C » . Elle consiste donc à construire d'abord les sous chemins optimaux et ensuite par récurrence le chemin optimal pour le problème entier. Cette méthode est destinée à résoudre des problèmes d'optimisation à vocation plus générale que la méthode de séparation et évaluation. Par contre, elle ne permet pas d'aborder des problèmes de taille aussi importante que le «branch and bound».

Programmation linéaire en nombres entiers

C'est l'une des techniques classiques de la recherche opérationnelle. C'est un cas particulier de la procédure de séparation et évaluation dont l'évaluation est basée sur une relaxation réelle. Cette méthode repose sur les travaux du simplexe et les algorithmes des points intérieurs de Karmarkar. Elle consiste à minimiser une fonction cout en respectant des contraintes. Le critére et les contraintes sont des fonctions linéaires des variables du problème .

Heuristiques

Les heuristiques sont des méthodes empiriques (qui donnent généralement de bons résultats sans être démontrables). Elles se basent sur des régles simplifées pour optimiser un ou plusieurs critéres. Le principe général de ces méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum. Nous exposons ci-dessous les plus utilisées d'entre elles .

FIFO (First In First Out) ou FCFS (First Come First Served) : La première tàche qui vient est la première tàche ordonnancée.

SPT (Shortest Processing Time) : La tàche ayant le temps opératoire le plus court est traitée en premier lieu.

LPT (Longest Processing Time) : La tàche ayant le temps opératoire le plus important est ordonnancée en premier lieu.

EDD (Earliest Due Date) : La tàche ayant la date due la plus petite est la plus prioritaire.

SRPT (Shortest Remaining Processing Time) : Cette régle sert à lancer la tàche ayant la plus courte durée de travail restant à exécuter. Elle est trés utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs.

ST (Slack Time) : A chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.

Métaheuristiques

Une méthode métaheuristique (ou, plus simplement, une métaheuristique) est une méthode approchée générique dont le principe de fonctionnement repose sur des mécanismes généraux indépendants de tout problème. Toutes les métaheuristiques s'appuient sur un équilibre entre l'intensification de la recherche et la diversification de celle-ci. D'un coté, l'intensification permet de rechercher des solutions de plus grande qualité en s'appuyant sur les solutions déja trouvées et de l'autre, la diversificationmet en place des stratégies qui permettent d'explorer un plus grand espace de solutions et d'échapper a des minima-locaux. Ne pas préserver cet équilibre conduit a une convergence trop rapide vers des minima-locaux (manque de diversification) ou a une exploration trop longue (manque d'intensification).

Méthodes de recherche locale

Les méthodes de recherche locale ou métaheuristiques a base de voisinages s'appuient toutes sur un méme principe. A partir d'une solution unique x_0, considérée comme point de départ (et calculée par exemple par une heuristique constructive), la recherche consiste a passer d'une solution a une solution voisine par déplacements successifs. L'ensemble des solutions que l'on peut atteindre a partir d'une solution x est appelé voisinage $N(x)$ de cette solution. Déterminer une solution voisine de x dépend bien entendu du problème traité. De maniére générale, les opérateurs de recherche locale s'arrétent quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Mais accepter uniquement ce type de solution n'est bien sur pas satisfaisant.

Méthodes de descente

A partir d'une solution trouvée par heuristique par exemple, on peut trés facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante. Cette méthode se base sur amélioration progressive de la solution et donc reste bloquée dans un minimum local dès qu'elle en rencontrere un. Il existe de maniére évidente une absence de diversification. Un moyen trés simple de diversifier la recherche peut consister à re-exécuter l'algorithmes en prenant un autre point de départ. Comme l'exécution de ces méthodes est souvent trés rapide, on peut alors inclure cette répétition au sein d'une boucle générale. On obtient alors un algorithme de type "Multi-start descent".

Recuit simulé

La méthode du recuit simulé a été introduite en 1983 par Kirkpatrick et al. Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire réguliére parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température est maintenu jusqu'à obtention d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires du métal non satisfaisants, mais conduisant à la longue à des états meilleurs. L'analogie avec une méthode d'optimisation est trouvée en associant une solution à un état du métal, son équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine. Pour passer à une solution voisine, il faut respecter l'une des deux conditions :

– soit le mouvement améliore la qualité de la solution précédente, i.e. En minimisation la variation de cout est négative ($DC < 0$),

– soit le mouvement détériore la qualité de la solution précédente et la probabilité p d'accepter un tel mouvement est inférieure à une valeur dépendant de la température courante t ($p < e^{-\Delta C/t}$)

Dans cet algorithme, l'équilibre entre intensification et diversification est respecté.

Recherche tabou

Contrairement au recuit simulé qui ne génére qu'une seule solution x_0 "aléatoirement" dans le voisinage $N(x)$ de la solution courante x, la méthode tabou, dans sa forme la plus simple, examine le voisinage $N(x)$ de la solution courante x. La nouvelle solution x_0 est la meilleure solution de ce voisinage (dont l'évaluation est parfois moins bonne que x elle-méme). Pour éviter de cycler, une liste tabou (qui a donné le nom à la méthode) est tenue à jour et interdit de revenir à des solutions déjà explorées. Dans une version plus avancée de la méthode tabou, on peut voir dans cette recherche une modification temporaire de la structure de voisinage de la solution x permettant de quitter des optima locaux. Le voisinage $N^*(x)$ intégrant ces modifications de structure est régit par l'utilisation de structures de mémoire spécifiques. Il s'agit de mémoire à court terme ou de mémoire à long terme. La mémoire à court terme correspond à la mise en place d'une liste tabou. La liste contient les quelques derniéres solutions qui ont été récemment visitées. Le nouveau voisinage $N(x)$ exclut donc toutes les solutions de la liste tabou. En conservant des caractéristiques des solutions ou des mouvements, il est possible alors qu'une solution de bien meilleure qualité ait un statut tabou. Accepter tout de méme cette solution revient à outrepasser son statut tabou, c'est l'application du critére d'aspiration. La mémoire à long terme permet d'une part d'éviter de rester dans une seule région de l'espace de recherche et d'autre part d'étendre la recherche vers des zones plus intéressantes. Dans cet algorithme, l'équilibre entre intensification et diversification est respecté.

Algorithmes génétiques

Cette classe de méthodes est basée sur une initiation des phénomènes d'adaptation des êtres vivants. L'application de ces méthodes aux problèmes d'optimisation a été formalisée par Goldberg en 1989. 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, décroisement 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 de 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 meilleurs. Toutefois, des solution de très mauvaise qualité 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 dont le nombre de paramètres à fixer (taille de la population, coefficients de reproduction, probabilité de mutation,...) de plus, ces algorithmes demandent un effort de calcul très important.

Chapitre 2

Notation et position du problème

Dans ce chapitre, nous proposons une étude du problème d'ordonancement $F2/recr(1)/Cmax$. Ce dernier est du type flow shop sur deux machines avec duplication de tâches sur la première 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.

2.1 Notation et position du problème

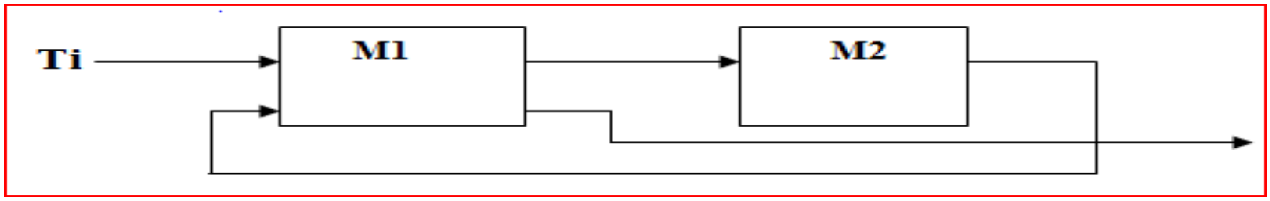
Le problème considéré est le suivant :

Soit $T = \{T1, T2, \ldots, Tn\}$ l'ensemble de n tâches indépendantes à ordonnacer sur un ensemble de deux machines $M = \{M1, M2\}$. L'atelier est de type flow shop. chaque tâche doit être excutée selon l'orde $M1 \rightarrow M2 \rightarrow M1$.

- chaque opération est exécutée sur une machine sans interruption (la préemption des tâches n'est pas autorisée),

- 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 :



Flow shop à deux machines avec recirculation

Pour un ordonnancement quelconque des tâches notons :

– p_{1i1} : la durée d'exécution de la première opération de la tâche Ti sur la première machine.

– p_{1i2} : la durée d'exécution de la deuxième opération de la tâche Ti sur la première machine.

– p_{2i} : la durée d'exécution de la tâche Ti sur la deuxième machine.

– s_{1i1} : la date de début d'exécution de la première opération de la tâche Ti sur la première machine.

– s_{1i2} : la date de début d'exécution de la deuxième opération de la tâche Ti sur la première machine.

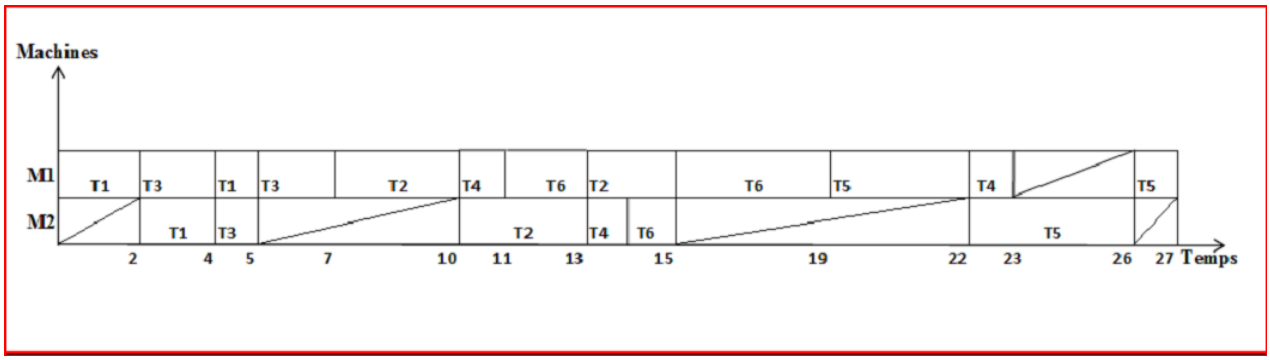
– s_{2i} : la date de début d'exécution de la tâche Ti sur la deuxième machine.

Exemple illustratif

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

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

Un ordonnancement est donné à la figure suivante :



2.2 Complexité du problème

M. Y. Wang, et al. dans leurs articles [28], "Minimizing makespan in a class of reentrant shops", considèrent le problème de flow shop à m machines avec recirculation de tâches sur la première machine $(M_1, M_2, \ldots., M_m, M_1)$ noté $Fm/chain - reentrant/C_{max}$ et ils proposent le théorème suivant :

Théorème 2.1 *(M. Y. Wang, et al.[28]) le problème $Fm/chain - reentrant/C_{max}$ est NP-difficile pour $m = 2$ et NP-dur pour $m \geq 3$.*

Remarque 2.1 *(M. Y. Wang, et al.[28]) pour $m = 2$ le problème $F2/recr(1)/C_{max}$ est équivalent à $V2//C_{max}$ introduit par Lev et Adiri (1984) qui est NP-difficile.*

donc notre problème est NP- difficile en général voir [28,19].

2.3 Motivations

Dans les problèmes d'ordonnancements classiques de type flow shop, nous supposons que chaque tâche visite chaque machine une seule fois. Mais ceci est souvent violé dans la pratique. Par exemple, dans la fabrication des semi-conducteurs, où chaque composant revisite la même machine plusieurs fois [27]. Cette caractéristique de recirculation peut être également trouvée dans la fabrication des cartes d'impression [28], on peut rencontrer aussi ce genre de problème dans les ateliers de fabrication de meubles ou chaque composant

subi une opération de peinture, puis une opération de polissage et enfin une opération de peinture.

2.4 Etat de l'art

Durant ces derniéres années, le problème de l'ordonnacement dans un flow shop a retenu l'attention de plusieurs chercheurs. Depuis que Johnson a proposé des solutions optimales pour des problèmes à deux ou trois machines, diverses heuristiques ont été développées pour résoudre le cas général. Le problème de l'ordonnacement dans un flow shop est un problème de production dans lequel n piéces doivent être exécutées suivant le méme ordre sur chacune des m machines qui composent l'atelier ; ces piéces ont donc toutes la méme gamme opératoire (l'ordre de passage sur les machines), mais pas les mémes temps d'exécution. La durée de traitement du produit i sur la machine j est p_{ij} $(i, j = 1, n)$. Dans la gamme opératoire des divers produits, si tous les temps d'exécution sont strictement positifs, l'atelier est du type pur flow shop. Sinon, s'il existe des temps d'exécution nuls (la piéce ne devant pas subir une opération sur une machine particuliére), on parle d'un flow shop généralisé. Indépendamment de cette classification, si la séquence des piéces est la méme sur toutes les machines (pas de dépacement autorisé), on parle d'un flow shop de permutation. L'objectif est de trouver une séquence des produits minimisant l'heure de sortie de la derniére piéce fabriquée (makespan). Ce problème est libellé :

$$Fm//Cmax$$

Les hypothéses principales pour ce problème sont les suivantes :

– un ensemble de n produits est disponible pour le traıtement à l'heure 0 (chaque produit est composé de m opérations et chaque opération requiert une machine différente) ;

– les temps de changement d'outils pour les opérations sont indépendants de la séquence et sont inclus dans le temps d'exécution ;

– la description des produits est connue à l'avance ;

– les m machines différentes sont disponibles en permanence (on suppose donc qu'il n' y pas de panne) ;

– les opérations sont non préemptives (c'est-à-dire que leur exécution se fait sans interruption sur les machines).

L'énumération compléte, les techniques de séparation et évaluation (Branch and Bound) ou la programmation en nombres entiers déterminent la séquence optimale pour des problèmes de petite taille mais des heuristiques sont nécessaires pour résoudre ceux de grande taille. En effet, le problème de flow shop considéré est NP-difficile lorsque le nombre de machines est supérieur ou égal à 3.

problème de Johnson

Un nombre non négligeable de problèmes d'ordonnancement sur une machine peuvent être résolus grâce à des méhodes exactes. Malheureusement, lorsque l'on étudie des problèmes à deux machines ou plus, il faut trés vite déchanter : un seul problème particulier à deux machines posséde une méthode de résolution exacte. Pour les autres, seules des heuristiques sont envisageables, comme nous le verrons plus loin. Ce problème particulier est le problème de Johnson, en voici l'énoncé : Un ensemble de n tâches doit être exécuté sur deux machines 1 et 2. il n'y a pas de place de stockage devant les machines. Chaque tâche doit passer d'abord dans la machine 1, puis dans la machine 2. Pour le reste, les cinq hypothéses de base sont respectées. En 1954, Johnson édicta une régle permettant de trouver une séquence optimale.

Régle de Johnson

La tâche i précéde la tâche j dans une séquence optimale si :

$$min\{p_{i1}, p_{j2}\} \leq min\{p_{j1}, p_{i2}\}.$$

Ainsi, suivant cette régle, le début de la séquence optimale est composée des tâches ayant un temps d'exécution court sur la première machine et la fin de la séquence contient les tâches ayant un temps d'exécution court sur la seconde machine. Johnson proposa ensuite un algorithme basé sur cette régle. L'algorithme de Johnson donne la séquence optimale pour le problème de Johnson. Cependant, ce problème est bien particulier, puisqu'il ne tient pas compte des délais. Rappelons que la détermination d'un ordonnancement (de durée minimale) des piéces sur une machine en respectant des dates de disponibilité et d'échéances est NP-difficile. La recherche d'heuristiques efficaces pour trouver une bonne méthode de résolution reste donc un domaine trés ouvert.

Algorithme de Johnson

L'algorithme de Johnson est fondé sur le calcul itératif du minimum des durées des tâches

associées aux travaux non encore placés. Cet algorithme donne une solution optimale pour le problème à deux machines de type flow shop. Nous présentons ci-dessous l'algorithme de Johnson.

Algorithme 2.1 *1 : $U = \emptyset; V = \emptyset$:*

2 : ***Tant que*** *la liste des tâches est non vide faire*

3 : Placer dans l'ensemble U les tâches pour lesquels $p_{j1} \leq p_{j2}$

4 : Placer dans l'ensemble V les tâches pour lesquels $p_{j1} > p_{j2}$

5 : ***Fin tant que***

6 : Ordonnancer les tâches de l'ensemble U dans l'ordre croissant des p_{j1}

7 : Ordonnancer les tâches de l'ensemble V dans l'ordre décroissant des p_{j2}

8 : Fusionner les deux ensembles U et V

9 : Exécuter la permutation obtenue sur les deux machines tout en respectant le méme ordre.

Flow shop de permutation à deux machines

Le problème de flow shop de permutation à deux machines $F2/prmu/Cmax$ est un problème de flow shop à deux machines pour lequel seules les séquences de permutation sont considérées. Le résultat le plus connu pour ce problème a été énoncé par Johnson [18]. Celui-ci a montré que toute séquence respectant « la régle de Johnson » est optimale, une séquence optimale unique de ce type pouvant être calculée en $O(nlogn)$. Sur la base de ce résultat, plusieurs auteurs se sont intéressés à déterminer un ensemble de solutions optimales pour le problème de flow shop de permutaion à deux machines $F2/prmu/Cmax$. Dans [12] est par exemple proposé un algorithme permettant d'énumérer toutes les séquences satisfaisant la régle de Johnson. Billaut et Lopez, dans [3], proposent aussi un algorithme qui énumére, en permutant des travaux 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 [3]. Pour les deux derniers cas, notons que seuls des problèmes de faible taille, d'une dizaine de travaux au plus, peuvent être résolus du fait de la complexité sous-jacente à l'énumération de toutes les solutions optimales.

problème de flowshop à contrainte « no-idle »

La contrainte « no-idle » signfie que dès qu'une machine est activée, elle ne doit pas s'arréter entre sa première et sa derniére utilisation. Le problème de l'ordonnancement flow-shop à contrainte « no-idle » et critére makespan ($noté\ F/no-idle/C_{max}$) est prouvé être NP-difficile. En 1997, Baptise et Hguny sont les premiers à avoir abordé le problème d'ordonnancement d'un flowshop de permutation avec contrainte no-idle (sans temps d'arrét) et critére makespan. Ils ont proposé un algorithme de type séparation et évaluation qui adopte un critére d'optimisation s'appuyant sur une hypothése restrictive.

problème de flowshop à contrainte « no-wait »

La contrainte sans délai « no-wait » signifie qu'aucune attente entre deux opérations d'une méme tàche n'est permise. Le problème $F2/no-wait/Cmax$ est résolu en temps polynomial par l'algorithme de Gilmore et Gomory. Ultérieurement, des contraintes additionelles ont été introduites comme la contrainte de blockage qui exprime qu'une tàche ne doit libérée une machine que pour passer directement sur la machine suivante par exemple le problème $F2/block(1,2)/C_{max}$ signfie qu'il existe une contrainte de blockage entre la première et la deuxième machine.

Cas à m $\geq$ 2 machines

Lorsque l'atelier est composé de deux machines ou plus, le cheminement des pièces détermine le type du problème. Si toutes les pièces ont la méme progression (chaîne de montage), on parle d'un probème de flow shop ; si cette restriction n'est pas présente, il s'agit d'un problème de job shop.

Flow shop généralisé de permutation :

Dans le cas de 3 machines, il existe une extension de l'algorithme de Johnson donnant une solution optimale si une condition fondamentale est remplie : la machine intermédiaire ne doit pas créer de bouchon, c'est-à-dire que les piéces se présentant devant cette machine sont servies immédiatement. De nombreuses heuristiques ont été développées depuis une trentaine d'années. La grande majorité d'entre elles utilisant des artifices de calcul pour se ramener au problème de Johnson à deux machines afin de pouvoir utiliser l'algorithme de Johnson. En 1965, Palmer a proposé un classement d'indice décroissant (slope index order)

pour ordonner les tâches sur les machines en fonction des temps d'exécution croissants dans leur gamme opératoire.

Algorithme 2.2 *Heuristique de Palmer :*

1 : ***Pour*** $i = 1$ *à* n *faire*

2 : calculer la valeur de l'indice S_i *tel que :*

3 : $S_i = (m-1)p_{im} + (m-3)p_{i,m-1} + (m-5)p_{i,m-2} + \ldots - (m-3)p_{i2}(m-1)p_{i1}$.

4 : ***Fin pour***

5 : la séquence est déterminée en classant les tâches par ordre décroissant des indices

6 : $S[1] \geq S[2] \geq \ldots \geq S[n]$

Cette méthode est légérement différente de la régle de Johnson et ne garantit pas l'optimalité, que l'on considére des problèmes à 2 machines ou plus. En 1971, Gupta a présenté une autre heuristique, trés similaire à celle de Palmer, à l'exception du calcul de l'indice. Il s'est apercu que la régle de Johnson donne une séquence optimale pour le problème à 3 machines en classant les tâches par ordre décroissant des indices S_i

$$S_i = e_i \ / \min \{p_{i1} + p_{i2}; p_{i2} + p_{i3}\}$$

$$e_i = \begin{cases} 1 \text{ , si } p_{i1} < p_{i3} \\ 0 \text{ , si } p_{i1} \geq p_{i3} \end{cases}$$

En généralisant au cas m $>$ 3 machines Gupta a proposé l'heuristique suivante :

Algorithme 2.3 *Heuristique de Gupta :*

1 : ***Pour*** $i = 1$ *à* n *faire*

2 : calculer la valeur de l'indice S_i *tel que :*

3 : $S_i = e_i / min \ \{p_{ik} + p_{i,k+1} : 1 \leq k \leq m-1\}$ *;*

4 : $(e_i = 1$ *; si* $p_{i1} < p_{im}$ *et* $e_i = -1$ *; si* $p_{i1} \geq p_{im})$

5 : ***Fin pour***

6 : La séquence est déterminée en classant les tâches par ordre décroissant des indices

7 : $S[1] \geq S[2] \geq \ldots \geq S[n]$

Gupta a comparé son heuristique à celle de Palmer pour de nombreux problèmes et a obtenu des résultats plus satisfaisants dans la grande majorité des cas.

2.4.1 Flow shop avec récirculation

Dans les problèmes de flow shop classique on suppose que chaque tâche visite chaque machine une seule fois mais cette contrainte est souvent violée dans la pratique, par exemple dans les ateliers de fabrication des semi conducteurs chaque tâche revisite les mêmes machines plusieurs fois. Récemment plusieurs articles traitent le problème de flow shop avec recirculation. Choi et Kim traite le problème de flow shop sur m machines avec recirculation tel que chaque tâche possède le process de fabrication ($M_1, M_2, \ldots, M_m$, $M_1, M_2, \ldots, M_m, \ldots, M_1, M_2, \ldots, M_m$) et considèrent le makispan comme critère d'optimisation ils ont développé plusieurs heuristiques effectives, Chen et al. propose un hybride génétique algorithme pour résoudre le même problème, Pan et Chen traitent le même problème pour lequel seules les séquences de permutation sont considérées, ils ont développé trois formulations mathématiques sous forme de programmes linéaires en variables mixtes et six heuristiques, Chen et al. adopte un hybride tabou search algorithme pour ce même problème, Choi et Kim considèrent le problème (M_1, M_2, M_1, M_2) il ont développé des propriété de dominances, des bornes inférieurs et des heuristiques et ils ont utilisé leurs résultats pour développer un algorithme de type séparation et évaluation (Branch and Bound) . Meziani Nadjat considére le problème de flow shop à deux machines avec recirculation des tâches sur la deuxième machines (M_1, M_2, M_2) , elle a prouvée que ce problème est polynomial et propose l'algorithme suivant pour le résoudre :

Algorithme F2/recr(2)/Cmax

Début

- Transformer le problème considéré au problème $F2//C_{max}$ comme suit :

- Sur la première machine, les temps de traitement ne changent pas $p\prime_{1i} = p_{1i}$;

- Sur la deuxième machine, les différentes opérations d'une même tâche forment une seuls opération tel que $p\prime_{2i} = p_{2il}$;

- Résoudre le problème $F2//C_{max}$ avec l'algorithme de Johnson.

- Construire la solution du problème initial $F2/recr(2)/C_{max}$ en partageant la durée de traitement de chaque tâche sur la deuxième machine en durées de traitement de ses opérations élémentaires initiales.

Fin

Présentons maintenant le principal travail donné par M. Y. Wang, et al.[28] .

On note : $x < y, x \leq y, x > y$ et $x \geq y$ si l'opération x précède y, x précède immédiatement y (pas de temps mort entre les deux opérations x et y), x suit y et x suit immédiatement y, respectivement.

Définition 2.1 *Un ordonnancement est compact sur une machine M_k si toutes les opérations O_{jk} $j = 1, \ldots, n$ sont ordonnancées ensembles sans qu'il n'y ait un temps mort entre les différentes opérations traitées sur cette machine c.-à-d $O_{ik} \leq O_{jk}$.*

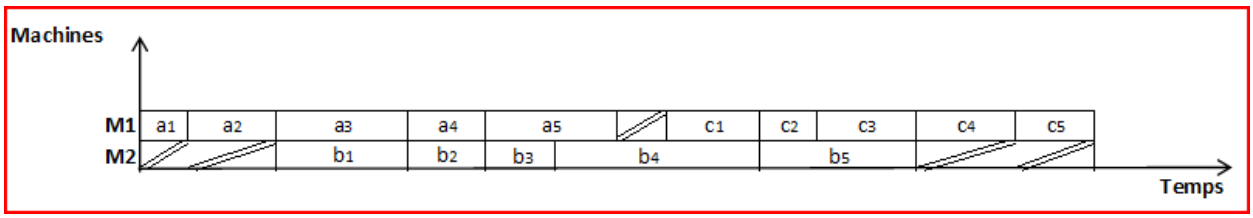
- Un ordonnancement est dit compact s'il est compact sur toutes les machines.

- Un ordonnancement est dit de permutation si l'ordre de traitement des tâches est le même sur les différentes machines.

Théorème 2.2 *(M. Y. Wang, et al.[28]) Pour le problème $Fm/chain-reentrant/C_{max}$, il suffit de considérer les ordonnancements compacts sur M_1 dont l'ordre de traitement des tâches est le même de M_1 à M_2 et de M_m à M_1.*

Corollaire 2.3 *(M. Y. Wang, et al.[28]) Pour le problème $F2/recr(1)/C_{max}$, il suffit de considérer les ordonnancements compacts et de permutations.*

D'après ce corollaire un ordonnancement compact et de permutation est constitué de 3 bloques d'opérations 1-bloque constitué uniquement des 1ères opérations (de la machine M1), 2-bloque constitué uniquement des 2èmes opérations (de la machine M_2) et 3-bloque constitué uniquement des 3èmes opérations (de la machine M1). Les opérations sont ordonnancées dans le même ordre sans aucun temps mort dans chaque bloque. Voir la figure ci-dessous ($a_j = p_{1j1}, b_j = p_{2j}, c_j = p_{1j2}$.)



Notons $\alpha = \sum_{j=1}^{n} P_{1j1}, \beta = \sum_{j=1}^{n} P_{2j}$ et $\gamma = \sum_{j=1}^{n} P_{1j2}$.

Soient :

- $C_1(A)$: la date de fin de traitement du premier passage des tâches de la séquence A sur la première machine.

- $C_2(A)$: la date de fin de traitement des tâches de la séquence A sur la deuxième machine.

- $C_3(A)$: la date de fin de traitement du deuxième passage des tâches de la séquence A sur la première machine.

Dans un ordonnancement compact de permutation on a :

1- il existe une tâche T_j tel que $s_{1j1} + a_j = s_{2j}$ et

2- si $C_{max} > \alpha + \gamma$ alors il existe T_j tel que $s_{2j} + b_j = s_{1j2}$.

Pour n'importe quel ordonnancement σ compact et de permutation, il existe une tâche $OT_{k^*} = arg\ \min_j\{s_{\sigma(j),2} + b_{\sigma(j),2} > \alpha\}$. D'où O_{k^*2} est la premiére 2-opération qui termine après α.

Donc l'ensemble ξ des tâches est partitionné en 3 sous ensembles $\xi_1 = \{T_j/T_j < T_{k^*}\}$ et $\xi_2 = \{T_j/T_j > T_{k^*}\}$ et T_{k^*} la tâche de partition.

Nous définissons une partition $\{\xi_1, T_{k^*}, \xi_2\}$ admissible si, et seulement si, elle satisfait

- $\xi_1 \cup \xi_2 \cup \{T_{k^*}\} = \xi$.

- $C_2(\xi_1) \leq \alpha$, et

- $C_2(\xi_1) + b_{k^*} > \alpha$.

Théorème 2.4 *(M. Y. Wang, et al.[28]) Il existe un ordonnancement optimal constitué d'une partition admissible $\{\xi_1, T_{k^*}, \xi_2\}$ telles que les tâches de ξ_1 sont ordonnancées selon la règle de Johnson pour (ξ_1, a, b) et les tâches de ξ_2 sont ordonnancées selon la règle de Johnson pour (ξ_2, b, c). L'ordonnancement obtenu est compact et de permutation.*

Des cas particuliers sont aussi présentés dans l'article M. Y. Wang, et al.[28], résolus en appliquant la règle de Johnson au pseudo problème $F2//Cmax$ qui est obtenu en posant $a'_j = a_j + b_j$ et $b'_j = b_j + c_j$.

1er cas Une machine domine les autres en terme de durées de traitements, c.-à-d., une des conditions suivantes est vérifiée :

– $min\{a_j\} \geq max\{b_j\}$.

- $min\{c_j\} \geq max\{b_j\}$.
- $min\{b_j\} \geq max\{a_j\}$.
- $min\{b_j\} \geq max\{c_j\}$.

2eme cas La deuxième machine est régressive

- $b_j < min\{a_j, c_j\}$.

3eme cas Les temps de traitement des opérations de la deuxième machine sont constants

- $b_j = b = constant$.

4eme cas $a = c$

Ce cas particulier est NP-difficile. M. Y. Wang, et al.[28] montrent que si $a_j = c_j$ pour tout tâches T_j, alors ce problème est résolu par un algorithme dynamique pseudo-polynômial en $O(n^2\alpha^2\beta)$.

Une méthode exacte de type séparation et évaluation et des heuristiques sont aussi présentés.

Séparation et évaluation algorithme M. Y. Wang, et al.[28]

En utilisant le théorème précédent M. Y. Wang, et al.[28], ont développés un algorithme de type séparation et évaluation qui énumère (implicitement) toutes les $n2^{n-1}$ partitions de ξ et trouve une partition minimisant le makespan. Au premier niveau de l'arbre nous indiquons les tâches de partitions réindexées et réordonnées selon la règle de Johnson pour les deux premières opérations ; en conséquence, il y a n nœuds dans ce niveau, puisqu'aucune tâche ne peut être éliminée. L'arbre est binaire au delà de ce niveau : à chacun des niveaux plus bas, nous décidons pour la tâche correspondante si elle sera ordonnancée avant ou après la tâche de partition. Un nœud au lième niveau ($l = 2, ..., n$) correspond alors à une partition partielle de ξ (A, T_{k^*}, B). Nous utilisons la stratégie en profondeur d'abords pour explorer l'arbre de recherche.

Un nœud peut être stérilisé s'il ne peut pas induire à une partition admissible. Une condition suffisante pour stériliser un nœud est donc $C_2(A) > \alpha$. En outre, si

$$max\left\{C_1(A) + a_{k^*}, C_2(A)\right\} + b_{k^*} > \alpha$$

alors la seule partition admissible que le nœud courant peut induire est $\{A, T_{k^*}, \xi \backslash A\}$ après l'évaluation , nous pouvons stériliser ce nœud.

Dans la racine de l'arbre nous calculons des bornes inférieures pour vérifier si la borne supérieure d'un candidat coïncide avec la valeur de la solution optimale :

Dans les autres nœuds de l'arbre, nous employons deux simples et efficaces bornes inférieures

$$C_2(A) + \sum_{T_j \in \xi \setminus A} b_j + \min_{T_j \in \xi \setminus A} c_j,$$

$$C_3(A) + \sum_{T_j \in \xi \setminus A} c_j.$$

Heuristiques M. Y. Wang, et al.[28]

Heuristique H1

1. Ordonnancer les tâches en appliquant la règle de Johnson à (ξ, a, b) pour obtenir l'ordre σ_1.

2. Ordonnancer les tâches en appliquant la règle de Johnson à (ξ, b, c) pour obtenir ordre σ_2.

3. $H1(I) = Min\{C_{max}(\sigma_1), C_{max}(\sigma_2)\}$.

Heuristique H2

1. Appliquer la règle de Johnson à (ξ, a, b) pour trouver l'ordre σ_1 ; comme dans l'heuristique $H1$, calculer $C_{max}(\sigma_1)$. Si $C_{max}(\sigma_1) = \alpha + \gamma$, arrêter.

2. Sinon identifier la tâche T_{k^*} et par conséquent la partition $\{\xi_1, T_{k^*}, \xi_2\}$.

3. Appliquer la règle de Johnson à $(T_{k^*} \cup \xi_2, b, c)$, tout en préservant l'ordre des tâches obtenu en ξ_1. Obtenir le nouveau ordre σ .

4. Ordonnancer les tâches selon σ et calculer $C_{max}(\sigma)$.

Heuristique H3

1. Appliquer la règle de Johnson à (ξ, b, c) pour trouver l'ordre σ_2 ; comme dans l'heuristique $H1$, calculer $C_{max}(\sigma_2)$. Si $C_{max}(\sigma_2) = \alpha + \gamma$, arrêter.

2. Sinon identifier la partition $\{\xi_3, T_{l^*}, \xi_4\}$ comme dans l'heuristique $H2$.

3. Appliquer la règle de Johnson à $(T_{l^*} \cup \xi_3, a, b)$, pour obtenir l'ordre final σ .

4. Ordonnancer les tâches selon σ et calculer $C_{max}(\sigma)$.

Notons par $\rho = \inf_{h \in H}\{r_h \ / \ r_h \geq C_{max}(S_h)/C_{max}(S^*)\}$ avec $r_h = \frac{C_{max}(S_h)}{borne\ inferieur}$ le rapport de performance, avec $C_{max}(S^*)$ est l'optimum et $C_{max}(SH)$ est la valeur de C_{max} donnée par l'heuristique h pour l'ordonnancement S.

On dit que le rapport de performance ρ est atteint s'il existe une instance du problème pour laquelle $C_{max}(SH)/C_{max}(S^*) = \rho$ ou bien $C_{max}(SH)/C_{max}(S^*) \rightarrow \rho$ quand les temps de traitement tendent vers zéro ou bien vers l'infini.

Théorème 2.5 *(M. Y. Wang, et al.[17]) Pour n'importe quelle instance $I = (\xi, a, b, c)$, ρ_1(de l'heuristique H1) est inférieur ou égal à $3/2$.*

Inna G. Drobouchevitch et al.[17] considèrent le même problème d'ordonnancement $F2/recr(1)/C_{max}$ et donnent un exemple pour lequel ρ_1 de l'heuristique H1 proposée par M. Y. Wang, et al.[28] est égale à $3/2$ et améliore ce dernier en proposant une heuristique qui garantie un rapport de performance inférieur à $4/3$. ci-dessous l'essentiel de leurs article.

Notons par :

$N = \{1, 2, ..., n\}$.

S^* : l'ordonnancement qui minimise le makespan.

$R_{Lj}(S)$: la date de début de traitement de l'opération O_{Lj} de la tâche j pour l'ordonnancement S.

$C_{Lj}(S)$: la date de fin de traitement de l'opération O_{Lj} de la tâche j pour l'ordonnancement S, avec $L \in \{A, B, C\}$ et $j = 1, 2, ..., n$.

Soit $Q \subset N$ un sous ensemble de N, on note par

$$a(Q) = \sum_{j \in Q} a_j, b(Q) = \sum_{j \in Q} b_j, c(Q) = \sum_{j \in Q} c_j,$$

Un ordonnancement S réalisable pour le problème $F2/recr(1)/C_{max}$ peut être associé à une permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$ telle que les opérations O_{Lj} sont traitées selon π pour tout $L \in \{A, B, C\}$ et chaque tâche commence le plutôt possible.

$$C_{\max}(S) = \max\{a(N) + c(N), \hat{C}(S)\}, \tag{1}$$

avec $\hat{C}(S) = \{\sum_{j=1}^{\mu} a_{\pi(j)} + \sum_{j=\mu}^{\nu} b_{\pi(j)} + \sum_{j=\nu}^{n} c_{\pi(j)} / 1 \leq \mu \leq \nu \leq n\}$.

Si $C_{\max}(S) = \hat{C}(S)$ et le maximum de (1) est atteint pour $\mu = u$ et $\nu = v$ alors $\pi(u)$ et $\pi(v)$ sont critiques dans S.

On dit que $\pi(u)$ est AB-critique dans S (elle est notée par $r_{AB}(S)$) et $\pi(v)$ est BC-critique dans S (elle est notée par $r_{BC}(S)$).

Notons aussi $\pi(w)$ la tâche de partition définie par M. Y. Wang, et al.[3], rappelons que $\pi(w)$ est choisit de telle sorte que $C_{B\pi(w-1)}(S) \leq a(N), C_{B\pi(w-1)}(S) > a(N)$.

Pour le problème $F2//C_{max}$ chaque tâche de l'ensemble $N = \{1, 2, ..., n\}$ est traitée sur M' puis sur M''. Les temps de traitements de M' et M'' sont notés par α_j, β_j respectivement. Pour une permutation π, on a la valeur du makespan égale à :

$$C^0_{\max}(\pi) = \max\{\sum_{j=1}^{\mu} \alpha_{\pi(j)} + \sum_{j=\mu}^{n} \beta_{\pi(j)} / 1 \leq \mu \leq n\}. \tag{2}$$

Si le maximum de (2) est atteint pour $\mu = u$ alors $\pi(u)$ est dite critique dans π.

$$N^\alpha = \{j \in N \backslash \alpha_j \leq \beta_j\}, N^\beta = \{j \in N \backslash \alpha_j > \beta_j\}. \tag{3}$$

Notons par **Algorithme1** et **Algorithme3,** l'algorithme de Johnson pour $F2//Cmax$ et l'heuristique H1 de M. Y. Wang, et al.[3], respectivement, et **Algorithme2,** l'algorithme de Gonzalez–Sahni pour le problème $O2//C_{max}$ ci-dessous

Algorithme2 Gonzalez–Sahni

1. Partitionner l'ensemble N en 2 sous-ensembles N^α et N^β selon (3)

2. Trouver les tâches $l \in N^\alpha$ et $r \in N^\beta$ telles que

$$\begin{aligned} \beta_l &= \max\{\beta_j / j \in N^\alpha\} \text{ Si } N^\alpha \neq \emptyset \\ \alpha_r &= \max\{\alpha_j / j \in N^\beta\} \text{ Si } N^\beta \neq \emptyset \end{aligned}$$

3. Former l'ordonnancement $\varphi = (\varphi_{(1)}, \varphi_{(2)}, ..., \varphi_{(n)}) = (l, \pi(N^\alpha|\{l\}), \pi(N^\beta)|\{r\}, r)$. telle que π est une permutation arbitraire.

Lemme 2.6 *(Inna G. Drobouchevitch et al.[17]) Pour le problème $F2//C_{max}$ la tâche l où bien r est critique pour l'ordonnancement φ.*

Soit S_W l'ordonnancement obtenu par l'algorithme3, on a $C_{max}(S_W)/C_{max}(S^*) \leq 3/2$. Inna G. Drobouchevitch et al.[17] montrent que ce rapport de performance maximum est atteint pour l'instance suivante :

	$T1$	$T2$	$T3$
$M1$	1	1	M
$M2$	M	M	1
$M1$	1	1	M

On a $C_{max}(S_W) = 3M + 2$ et $C_{max}(S^*) = 2M + 4$ donc $C_{max}(S_W)/C_{max}(S^*) = \frac{3M+2}{2M+4} \rightarrow \frac{3}{2}$.tel que $M >> 1$.

Présentons les inégalités suivantes :

$$1/3C_{max}(S^*) < a(N) < 2/3C_{max}(S^*) \tag{4}$$

$$1/3C_{max}(S^*) < c(N) < 2/3C_{max}(S^*) \tag{5}$$

Lemme 2.7 *(Inna G. Drobouchevitch et al.[17]) Soit S_W un ordonnancement obtenu par l'algorithme3. Si une des inégalités* (4) *et* (5) *n'est pas vérifiée, alors $C_{max}(S_W)/C_{max}(S^*) \leq 4/3$.*

Supposons alors que les deux inégalités (4) et (5) ne sont pas vérifiées, une tâche k est dite grande si elle vérifie les deux inégalités suivantes :

$$a_k + b_k \geq 2/3C_{max}(S^*) \tag{6}$$

$$b_k + c_k \geq 2/3C_{max}(S^*) \tag{7}$$

Définissons les ensembles N_k^a et N_k^c pour une tâche k tel que $1 \leq k \leq n$

$$N_k^a = \{j \in N | \{k\}/a_j \leq c_j\} \text{ et } N_k^c = \{j \in N | \{k\}/a_j > c_j\}. \tag{8}$$

En utilisant (8), on forme la permutation Ψ_k ci-dessous

$$\Psi_k = (\pi(N_k^a), k, \pi(N_k^c)), \tag{9}$$

telle que $\pi(Q)$ est une permutation arbitraire de Q.

Lemme 2.8 *(Inna G. Drobouchevitch et al.[17]) Soit une tâche k tel que $1 \leq k \leq n$ qui satisfait* (6) *et* (7) *pour un ordonnancement S_Ψ définie par la permutation Ψ_k , alors* $C_{max}(S_\Psi)/C_{max}(S^*) \leq 4/3$.

Nous présentons maintenant l'heuristique de Inna G. Drobouchevitch et al.[17] . Cette dernière crée plusieurs ordonnancement et choisit le meilleur.

Heuristique (Inna G. Drobouchevitch et al.[17])

1. Renuméroter les tâches de l'ensemble N selon l'ordre de Johnson $\sigma_{AB} = (1, ..., n)$.

2. Pour le problème $F2/recr(1)/C_{max}$, trouvez un ordonnancement S_0 lié à la permutation σ_{AB}.

Trouver la tâche w de partition telle que $C_{B\pi(w-1)}(S) \leq a(N), C_{B\pi(w-1)}(S) > a(N)$.

Définissons les ensembles de tâche $N_1 = \{1, 2, ..., w-1\}$ et $N_2 = \{w+1, w+2, ..., n\}$.

3. Appliquer l'algorithme 2 avec $\alpha_j = b_j$ et $\beta_j = c_j$ aux tâches de l'ensemble N_2 pour trouver une permutation de Gonzalez-Sahni $\varphi_{BC}(N_2) = (\varphi_{(1)}, \varphi_{(2)}, ..., \varphi_{(q)})$ avec $q = |N_2| = n - w$.

4. Pour $F2/recr(1)/C_{max}$ trouvez les ordonnancements liés aux permutations spécifiques des tâches comme indiqué ci-dessous :

$$
\begin{array}{ll}
S_1 & \sigma_{BC}(N) \\
S_2 & (\pi(N^a_{\varphi_{(1)}}), \varphi(1), \pi((N^c_{\varphi_{(1)}})) \\
S_3 & (\varphi(1), \sigma_{BC}(N) \backslash \{\varphi(1)\}) \\
S_4 & (\pi(N^a_{\varphi_{(q)}}), \varphi(q), \pi((N^c_{\varphi_{(q)}})) \\
S_5 & (\varphi(q), \sigma_{BC}(N) \backslash \{\varphi(q)\}) \\
S_6 & (1, 2, ..., w, \varphi_{BC}(N_2)) \\
S_7 & (w, \varphi_{BC}(N_2), 1, 2, ..., w-1) \\
S_8 & (\pi(N^a_w), w, \pi(N^c_w)) \\
S_9 & (\sigma_{AB}(N \backslash \{w\}), w) \\
S_{10} & (1, 2, ..., w-1, \varphi_{BC}(N_2), w) \\
S_{11} & (\varphi(2), \varphi(3), ..., \varphi(q), 1, 2, ..., w-1, w, \varphi(1))
\end{array}
$$

5. Parmi tous les ordonnancements déterminer celui qui minimise le makespan .

Cette heuristique est en $O(nlogn)$.

Théorème 2.9 *(Inna G. Drobouchevitch et al.[17]) Soit S_H un ordonnancement pour le problème $F2/recr(1)/Cmax$ trouvé par l'heuristique ci-dessus alors $C_{max}(S_H)/C_{max}(S^*) \leq 4/3$ et cette borne est atteinte.*

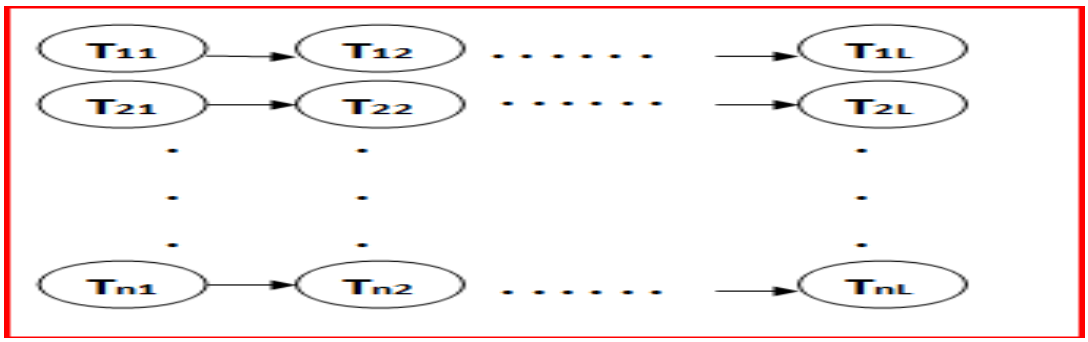
Ce rapport de performance maximum est atteint pour l'exemple suivant :

	$T1$	$T2$	$T3$	$T4$
$M1$	1	2	3	M
$M2$	M	6	M	M
$M1$	$M-2$	1	4	$M-1$

(Exemple)

On a $C_{max}(S_H) = 4M + 1$ et $C_{max}(S^*) = 3M + 8$ donc $C_{max}(S_H)/C_{max}(S^*) = \frac{4M+1}{3M+8} \rightarrow \frac{4}{3}$.tel que $M >> 1$.

Caixia Jing et al. [5] dans leur article considèrent le problème d'ordonnancement sur deux machines avec recirculation tel que chaque tâche se traite selon le process de fabrication suivant : $(M_1M_2, M_1M_2, ..., M_1M_2)$, chaque tâche peut être décomposée en L sous tâches telle que chaque sous tâche T_{ij} $i = 1..n$, $j = 1..L$ se traite sur les deux machines (M_1, M_2). Donc, on a L sous tâches avec contraintes de précédence qui signifie que le traitement de la $(l+1)^{ieme}$ sous tâche sur la première machine M_1 ne peut pas commencée avant que le traitement de la l'^{ieme} sous tâche sur la deuxième machine M_2 s'achève. De cette manière, on transforme le problème de flow shop avec recirculation en un problème de flow shop avec contraintes de précédences sous forme de chaines parallèles comme le montre le schéma suivant :



flow shop avec contraintes de précédences sous forme de chaines parallèles

Notons par :

n : le nombres de tâches à traiter.

T_i : la tâches numéro i.

L : le nombre de sous tâches de T_i

P_{ijl} : le temps de traitement de la lième sous tâche de T_i sur la machine j

C^*_{max} : minimum du makespan ou bien la valeur optimale de la fonction objectif.

Les contraintes de précédences sous formes de chaines parallèles sont un cas particulier des contraintes de précédences générales, on note $F2/chains/Cmax$ et $F2/prec/Cmax$ le problème flow shop à deux machines avec contraintes de précédences sous formes de chaînes parallèles et le problème de flow shop à deux machines avec contraintes de précédences respectivement.

Si on pose $L = 2$ et les temps de traitements de la deuxième machine égale à zéro pour $l = 2$ on aura exactement notre problème $(F2/recr(1)/Cmax)$.

La relation entre $F2/prec/Cmax$ et le problème d'ordonnancement sur une seule machine sous contraintes de précédences avec retard $1/prec(l_{ij}), p_j/Cmax$.

La description du problème d'ordonnancement sur une seule machine sous contraintes de précédences avec retard est comme suit : on a une seule machine et un ensemble de n tâches dépendentes des contraintes de précédences sous forme d'un graphe $G = (V, E)$ qui ne contient pas de cycle, tel que V est l'ensemble des sommets correspondant aux tâches et E est l'ensmble des arcs correspond aux contraintes de précédences. Ce graphe est un graphe pondéré tel que les sommets sont pondérés par p_i (temps de traitements de la tâche i , $i \in V$) et les arcs sont pondérés par l_{ij} (retard entre la tâche j et la tâche i) c.-à-d. la tâche j ne peut pas commencer qu'après l_{ij} unité de temps après la fin de la tâche i donc $C_i + l_{ij} \leq \sigma(j)$ avec :

- Ci date de fin de traitement de la tâche i.

- $\sigma(j)$ début de traitement de la tâche j.

l_{ij} et p_i sont des nombres entiers positifs. La fonction objectif c'est de minimiser la durée totale d'exécution (minimum du makespan), ce problème est noté par $1/prec(l_{ij}), p_j/Cmax$.

Théorème 2.10 *(Caixia Jing et al. [5]) pour le problème* $1/prec(lij), pj/Cmax$ *si*

1. $l_{ij} = l_i.i, j = 1, \ldots, n,$

2. $\forall k \neq i; p_k \geq l_i, i, j = 1, \ldots, n,$

alors le problème précédent est équivalent au problème flow shop sur deux machines sous contraintes de précédences $F2/prec/Cmax$ avec

$p_{j1} = p_j,\ j = 1, ..., n,$

$$p_{j2} = \begin{cases} l_j \ si \ l_j > 0, \\ 0 \ sinon. \end{cases}$$

Preuve. La seule déférence entre les délais de retard dans $1/Prec(l_{ij}), p_j/Cmax$ et les temps de traitement de la deuxième machine dans $F2/prec/Cmax$ est que les délais de retard peuvent être écoulés simultanément alors que les temps de traitements ne peuvent pas. Cependant la condition $\forall k \neq i; p_k \geq li, i, j = 1, \ldots, n.$ évite les chevauchements entre les tâches et la condition $l_{ij} = l_i.i, j = 1, \ldots, n.$ assure l'unicité des temps de traitements sur la deuxième machine. □

Finta et al. [13] montrent que le problème $1/prec(l_{ij} = 1), p_j \in N^+/Cmax$ lorsque les délais de retards sont unitaires et les temps de traitements sont entières est polynomial et peut être résolu en $O(n^2)$ en ordonnancant les tâches selon l'ordre lexicographique.

Théorème 2.11 *(Caixia Jing et al. [5]) Pour le problème* $F2/prec/Cmax$, *si*

1. $p_{j1} \in N^+,$

2. $p_{j2} = 1$ pour les tâches j qui ont au moins un successeur et $p_{j2} = 0$ pour les autres tâches.

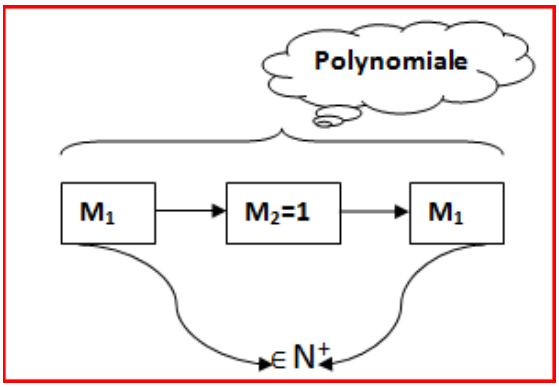
Alors le problème est polynomial et est résolu en $O(n^2)$ en ordonnançant les tâches selon l'ordre lexicographique.

Corollaire 2.12 *Caixia Jing et al. [5] Le problème* $F2/chains/Cmax$, *si* $p_{i1l} \in N^+$ *et* $p_{i2l} = 1$ *pour* $i = 1, ..., n$ *;* $l = 1, \ldots, L-1$ *et* $p_{i2l} = 0$ *pour* $l = L$ *alors le problème est polynomial et est résolu en* $O(n^2)$ *par l'algorithme lexicographique .*

Preuve. À partir du problème $F2/prec/Cmax$, dans lequel $p_{j1} \in N^+$ et $p_{j2} = 1$ pour les tâches j qui ont au moins un successeur et $p_{j2} = 0$ pour les autres tâches, on peut

construire le problème $1/prec(l_{ij}=1), p_j \in N^+/Cmax$ avec $p_j = p_{j1} \in N^+,\ j = 1, ..., n.$ et $l_j = p_{j2} = 1$; et comme ce dernier peut être résolu en $O(n^2)$ d'après Finta et al., alors on a la démonstration du théorème ci-dessus . □

Donc, si on pose $L = 2$, notre problème est polynomial sous ces conditions.



Théorème 2.13 *(Caixia Jing et al. [5]) Pour le problème $F2/chains/Cmax$. Si $\forall l, m$ et $k \neq i$ $p_{k1l} \geq p_{k2m}$ alors il existe une séquence optimale S des n tâches dont laquelle la dernière tâche à ordonnancer est la tâche h qui satisfait $P_{h2l} = \min_{1\leq i\leq n}\{P_{i2l}\}$ et les autres tâches sont positionnées arbitrairement. En ordonnançant les tâches selon S on obtient* $C_{max} = \sum_{1\leq i\leq n, 1\leq l\leq L} P_{i1l} + p_{h2L}$.

Théorème 2.14 *(Caixia Jing et al. [5]) Pour le problème $F2/chains/Cmax$. Si $\forall l, m$ et $k \neq i$ $p_{k1l} \leq p_{i2m}$ alors il existe une séquence optimale S des n tâches dont laquelle la dernière tâche à ordonnancer est la tâche h qui satisfait $P_{h11} = \min_{1\leq i\leq n}\{P_{i11}\}$ et les autres tâches sont positionnées arbitrairement. En ordonnançant les tâches selon S on obtient* $C_{max} = \sum_{1\leq i\leq n, 1\leq l\leq L} p_{i2l} + p_{h11}$.

Chapitre 3

Méthodes de résolutions

Dans ce chapitre, nous proposons quelques méthodes de résolution du problème d'ordonancement $F2/recr(1)/C_{max}$. Nous commençons par une modélisation du problème en programme linéaire en variables réelles et bivalentes. Nous proposons également des bornes inférieures et des sous problèmes polynomiaux avec des algorithmes pour les résoudre.

3.1 Modélisation mathématique

Soient les variables binaires $\alpha_{ii'jj'}$, β_{ij} définies par :

$$\alpha_{iji'j'} = \begin{cases} 1 \text{ si } s_{1ij} \leq s_{1i'j'} \\ 0 \text{ sinon .} \end{cases} \quad \forall i, i' = 1, \ldots, n. j, j' = 1, 2. \quad \text{avec } (i \neq i') \text{ ou } (i = i' \; et \; j \neq j').$$

$$\beta_{ii'} = \begin{cases} 1 \text{ si } s_{2i} \leq s_{2i'} \\ 0 \text{ sinon .} \end{cases} \quad \forall i, i' = 1, \ldots, n.$$

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

- Pour tout couple de tâches (T_i, T_j) nous avons :

$\alpha_{iji'j'} + \alpha_{i'j'ij} = 1 \quad i, i' = 1, \ldots, n. j, j' = 1, 2.$ avec $(i \neq i')$ ou $(i = i' \; et \; j \neq j')$.

$\beta_{ij} + \beta_{ji} = 1 \quad i, j = 1, \ldots, n \quad i \neq j.$

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

$s_{1ij} + p_{1ij} - s_{1i'j'} \leq A(1 - \alpha_{iji'j'}) \quad i, i' = 1, \ldots, n. j, j' = 1, 2.$ avec $(i \neq i')$ ou $(i = i' \; et \; j \neq j')$.

$s_{1i'j'} + p_{1i'j'} - s_{1ij} \leq A\alpha_{iji'j'} \quad i, i' = 1, \ldots, n.j, j' = 1, 2.$ avec $(i \neq i')$ ou $(i = i'$ *et* $j \neq j')$.

$s_{2i} + p_{2i} - s_{2j} \leq A(1 - \beta_{ij}) \quad i, j = 1, \ldots, n \quad i \neq j.$

$s_{2j} + p_{2j} - s_{2i} \leq A\beta_{ij} \quad i, j = 1, \ldots, n \quad i \neq j.$

Le traitement d'une tâche ne peut commencer sur la deuxième machine que si le traitement de sa première opération sur la première machine est achevé

$s_{1i1} + p_{1i1} \leq s_{2i}. \quad i = 1, \ldots, n.$

Le traitement de la deuxième opération d'une tâche ne peut commencer sur la première machine que si son traitement sur la deuxième machine est achevé

$s_{2i} + p_{2i} \leq s_{1i2}. \quad i = 1, \ldots, n.$

La date de fin de traitement de la deuxième opération d'une tâche sur la première machine est inférieur au makespan.

$s_{1i2} + p_{1i2} \leq y. \quad i = 1, \ldots, n.$

Les contraintes de positivité et de binarité

$$\begin{cases} \alpha_{iji'j'} \in \{0, 1\} \quad i, i' = 1, \ldots, n.j, j' = 1, 2.\text{avec } (i \neq i') \text{ ou } (i = i' et j \neq j'). \\ \beta_{ij} \in \{0, 1\} \quad i, j = 1, \ldots, n \quad i \neq j. \\ s_{1i1} \geq 0 \qquad\qquad i = 1, \ldots, n \\ s_{1i2} \geq 0 \qquad\qquad i = 1, \ldots, n \\ s_{2i} \geq 0 \qquad\qquad i = 1, \ldots, n \end{cases}$$

La fonction objectif

$C_{max} = min(y).$

Le nombre de variables et le nombre de contraintes d'un modèle mathématique linéaires 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 $(5n^2)$ et le nombres de contraintes est égale à $(15n^2 + 2n)$. Le but de l'expérience est d'avoir une idée sur la capacité de la résolution du 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 : si le nombre de tâches est égale à 10. Le nombre de variables est égale à 500 et le nombre de contraintes est égales à 1520. Nous remarquons que pour cet exemple de petite taille, le nombre de variables et de contraintes est très important.

Le modéle final

$$\begin{cases}
\mathbf{min(y)} \\
\alpha_{iji'j'} + \alpha_{i'j'ij} = 1 \quad i,i' = 1,\ldots,n.j,j' = 1,2. \quad \text{avec } (i \neq i') ou (i = i' et j \neq j'). \\
\beta_{ij} + \beta_{ji} = 1 \quad i,j = 1,\ldots,n \quad i \neq j. \\
s_{1ij} + p_{1ij} - s_{1i'j'} \leq A(1 - \alpha_{iji'j'}) \quad i,i' = 1,\ldots,n.j,j' = 1,2. \quad \text{avec } (i \neq i') \text{ ou } (i = i' et j \neq j'). \\
s_{1i'j'} + p_{1i'j'} - s_{1ij} \leq A\alpha_{iji'j'} \quad i,i' = 1,\ldots,n.j,j' = 1,2. \quad \text{avec } (i \neq i') \text{ ou } (i = i' et j \neq j'). \\
s_{2i} + p_{2i} - s_{2j} \leq A(1 - \beta_{ij}) \quad i,j = 1,\ldots,n \quad i \neq j. \\
s_{2j} + p_{2j} - s_{2i} \leq A\beta_{ij} \quad i,j = 1,\ldots,n \quad i \neq j. \\
s_{1i1} + p_{1i1} \leq s_{2i}. \quad i = 1,\ldots,n. \\
s_{2i} + p_{2i} \leq s_{1i2}. \quad i = 1,\ldots,n. \\
s_{1i2} + p_{1i2} \leq y. \quad i = 1,\ldots,n. \\
\alpha_{iji'j'} \in \{0,1\} \quad i,i' = 1,\ldots,n.j,j' = 1,2. \text{avec } (i \neq i') \text{ ou } (i = i' et j \neq j'). \\
\beta_{ij} \in \{0,1\} \quad i,j = 1,\ldots,n \quad i \neq j. \\
s_{1i1} \geq 0 \qquad\qquad i = 1,\ldots,n \\
s_{1i2} \geq 0 \qquad\qquad i = 1,\ldots,n \\
s_{2i} \geq 0 \qquad\qquad i = 1,\ldots,n
\end{cases}$$

Les modèles linéaires en variables entières et bivalentes peuvent être résolus par des solveurs efficaces tels que LINGO, CPLEX, etc.

3.2 Bornes inférieures

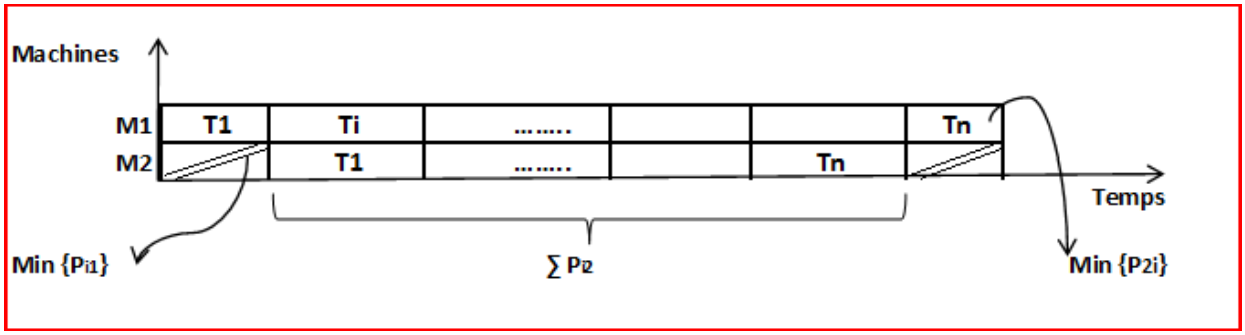
Remarque 3.1 $LB1 = \sum_{i=1}^{n}(p_{1i1} + p_{1i2})$ *est une borne inferieure.*

Preuve. Chaque tâche est exécutée deux fois sur la première machine, donc le temps de traitement de l'ensemble des tâches sur la première machine est égale à $\sum_{i=1}^{n}(p_{1i1} + p_{1i2})$, d'où LB1 est une borne inférieure. □

Remarque 3.2 $LB2 = \min_{i=1,\ldots,n}\{p_{1i1}\} + \sum_{i=1}^{n} p_{i2} + \min_{i=1,\ldots,n}\{p_{1i2}\}$ *est une borne inferieure.*

Preuve. Chaque tâche est exécutée une seule fois sur la deuxième machine, donc le temps de traitement sur la 2éme machine est égale à $\sum_{i=1}^{n} p_{i2}$, d'où $\sum_{i=1}^{n} p_{i2}$ est une borne inferieure, et comme une tâche ne peut pas être traitée sur la 2éme machine que si elle est déjà traitée

sur la 1ere machine ; et elle doit être traitée sur la 1er machine une second fois donc on élargit la borne inferieure en ajoutant les temps d'attentes. Finalement, LB2 est une borne inférieure (voir la figure ci-dessous).



□

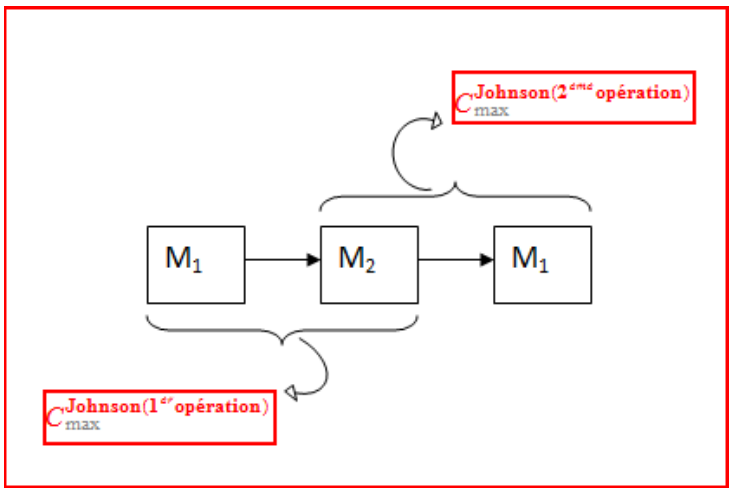
Remarque 3.3 $LB3 = Max\left\{C_{\max}^{\textbf{Johnson}(\textbf{1}^{er}\textbf{opération})} + \min_{i=1,\ldots,n}\{P_{1i2}\}\right\}$ *est une borne inferieure.*

où $C_{\max}^{\textbf{Johnson}(\textbf{1}^{er}\textbf{opération})}$ est la valeur du C_{max} donnée par l'algorithme de Johnson appliqué sur les premières opérations de la première machine et les opérations de la deuxième machine.

$C_{\max}^{\textbf{Johnson}(\textbf{2}^{eme}\textbf{opération})}$ est la valeur du C_{max} donnée par l'algorithme de Johnson appliqué sur les opérations de la deuxième machine et les deuxièmes opérations de la première machin en considérant M2 comme la première machine. .

Remarque 3.4 $LB4 = Max\left\{C_{\max}^{\textbf{Johnson}(\textbf{2}^{eme}\textbf{opération})} + \min_{i=1,\ldots,n}\{P_{1i1}\}\right\}$ *est une borne inferieure.*

Preuves évidentes .



Remarque 3.5 *$LB5 = Max(LB1, LB2, LB3, LB4)$ est une borne inferieure*

3.3 Sous problèmes polynomiaux

Corollaire 3.1 *Les sous-problèmes suivants sont résolus en temps polynomial :*

p_{1i1}	p_{2i}	p_{12i}	Complexité de l'algorithme de résolution
1	N^+	N^+	$O(nlogn)$, ordre de Johnson
1	1	N^+	$O(n)$, ordre quelconque
p	p	p	$O(n)$, ordre quelconque
N^+	1	1	$O(n)$, ordre quelconque
N^+	N^+	1	$O(nlogn)$, ordre de Johnson
1	N^+	1	$O(n)$, ordre quelconque
N^+	1	N^+	$O(nlogn)$, ordre de Johnson

Preuve. D'aprs l'article de M. Y. Wang, et al.[28] si une machine domine les autres en terme de durées de traitements, c.-à-d., une des conditions suivantes est vérifiée : $min\{a_j\} \geq max\{b_j\}, min\{c_j\} \geq max\{b_j\}, min\{b_j\} \geq max\{a_j\}, min\{b_j\} \geq max\{c_j\}$.alors le problème est polynomial, d'où on peut déduire que les cas particuliers ci dessus sont aussi polynomiaux. □

3.4 Heuristiques

L'heuristique H1 présente le squelette d'une méthode de descente simple. A partir d'une solution initiale x, on choisit une solution x_0 dans le voisinage $N(x)$ de x. Si cette solution est meilleure que x, $(f(x_0) < f(x))$ alors on accepte cette solution comme nouvelle solution et on recommence le processus jusqu' a ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de x.

Heuristique H1 Simple descent

1 : initialisation : trouver une solution initiale x

2 : **répéter**

3 : recherche voisinage : trouver une solution $x_0 \in N(x)$

4 : **Si** $f(x_0) < f(x)$ **alors**

5 : $x_0 = x$

6 : **FinSi**

7 : **jusqu'a** $f(y) \geq f(x), \forall y \in N(x)$.

Cette méthode se base sur une amélioration progressive de la solution et donc reste bloquée dans un minimum local des qu'elle le rencontre. Il existe de manière évidente une absence de diversification. L'équilibre souhaité entre intensification et diversification n'existe donc pas. Un moyen très simple de diversifier la recherche peut consister à ré-exécuter cet algorithme en prenant un autre point de départ. Comme l'exécution de cette méthode est souvent très rapide, on peut alors inclure cette répétition au sein d'une boucle générale. On obtient alors un algorithme de type Multi-start descent décrit par l'heuristique H2.

Heuristique H2 Multi-start descent

1 : initialisation : trouver une solution initiale x, $k = 1$, $f(B) = +\infty$.

2 : **répéter**

3 : point de départ : choisir une première solution x_0 au hasard

4 : $x =$(résultats d'une Simple Descente).

5 : **Si** $f(x) < f(B)$ **alors**

6 : $B = x$

7 : **finsi**

8 : $k = k + 1$

9 : **jusqu'a** satisfaire un critère d'arrêt.

Heuristique H3 Johnson etendue

1. Ordonnancer les premières opérations de l'ensemble des tâches sur les deux premières machines (M1, M2) selon la règle de Johnson.

2. Ordonnancer les deuxièmes opérations de l'ensemble des tâches dans le même ordre que 1 sur la première machine M1.

L'algorithme de la borne inférieur modifié (Modified lower bound-based algorithm (MLBB))

Ces deux dernières heuristiques sont présentées dont l'article de Caixia Jing et Al.[6] dans leurs articles ils considèrent le problème d'ordonnancement sur deux machines avec recirculation tel que chaque tâche se traite selon le process de fabrication suivant :

$(M_1M_2, M_1M_2, ..., M_1M_2)$ chaque tâche peut être décomposée en L sous tâches telle que chaque sous tâche T_{ij} $i = 1..n$, $j = 1..L$ se traite sur les deux machines (M_1, M_2) donc on a L sous tâches avec contraintes de précédence qui signifié que le traitement de $(l+1)^{ieme}$ sous tâche sur la première machine M_1 ne peut pas commencer avant que le traitement de la l'^{ieme} sous tâche sur la deuxième machine M_2 ne s'achève

Dans cet algorithme les sous tâches sont ordonnancées une par une selon la borne inferieure. Notons σ et U l'ordonnancement partiale courant et l'ensemble de tous les sous tâches non encore ordonnancées respectivement. les sous tâches disponibles sont touts les $l^{ième}$ sous tâches non encore ordonnancées sachant que les $(l-1)^{ième}$ sous tâches sont ordonnancées $l = 1, 2, ..., L$. Soit A l'ensemble de toutes les sous tâches disponibles. Évidemment A est un sous ensemble de U. À chaque itération de l'algorithme on choisit une sous tâche k de A qui vérifiée le minimum de la borne inferieure suivante :

$$B(\sigma k) = max \left\{ C_1(\sigma k) + C^J \max(U - \{k\}), C_2(\sigma k) + \sum_{a \in U - \{k\}} p_{a2} \right\}.$$

Heuristique H4

0 : l'ensemble $\sigma = \varnothing, C_1(\sigma) = C_2(\sigma) = 0$, soit U l'ensemble de toutes les sous tâches et A l'ensemble de toutes les sous tâches disponibles en première étape .

1 : si $U = \varnothing$ Terminer, Sinon pour toute sous tâche $k \in A$, calculer la borne inférieure $B(\sigma k)$.

2 : sélectionner une sous tâche k^* de A minimum des bornes inférieures.

$3 : \sigma \leftarrow \sigma k^*$, supprimer k^* de U et A, et ajouter $k^{*\prime}s$ le successeur immédiat de A calculer

$C1 = max\{C_1(\sigma), C_p(k^*)\} + p^*_{k1}$

$C2 = max\{C_1(\sigma), C_2(\sigma)\} + p^*_{k2}$

avec $C_p(k^*)$ le temps de fin de traitement de $k^{*\prime}s$ qui est le prédécesseurs immédiat de la 2éme machine. Aller à 1.

Si on pose $L = 2$ et les temps de traitements de la deuxième machine égale à zéro pour $l = 2$, on aura exactement notre problème.

L'algorithme des temps morts pondérés (Weighted idle time-based algorithm (WITB))

Heuristique H5

0 : l'ensemble $\sigma = \varnothing, C_1(\sigma) = C_2(\sigma) = 0$, calculer les facteurs de la 1^{er} machine et de la $2^{iéme}$ machine :

$$w_1 = max\left\{\frac{\sum_{i=1}^{nL} p_{i1}}{\sum_{i=1}^{nL} p_{i2}}, 1\right\}, \quad w_2 = max\left\{\frac{\sum_{i=1}^{nL} p_{i2}}{\sum_{i=1}^{nL} p_{i1}}, 1\right\}.$$

Soit U l'ensemble de toutes les sous tâches et A l'ensemble de toutes les sous tâches disponibles en première étape .

1 : si $U = \varnothing$, Terminer, Sinon pour toute sous tâche $k \in A$, calculer la somme des temps morts

$$I(\sigma k) = w_1(C_1(\sigma k) - C_1(\sigma) - p_{k1}) + w_2(C_2(\sigma k) - C_2(\sigma) - p_{k2}).$$

2 : sélectionner une sous tâche k^* de A minimum des temps morts.

$3 : \sigma \leftarrow \sigma k^*$ supprimer k^* de U et A, et ajouter $k^{*\prime}s$ le successeur immédiat de A calculer

$C1 = max\{C_1(\sigma), C_p(k^*)\} + p^*_{k1}$

$C2 = max\{C_1(\sigma), C_2(\sigma)\} + p^*_{k2}$

avec $C_p(k^*)$ temps de fin de traitement de $k^{*\prime}s$ le prédécesseurs immédiat de la 2éme machine. Aller à 1.

Si on pose $L = 2$ et les temps de traitements de la deuxième machine égale à zéro pour $l = 2$, on aura exactement notre problème .

3.5 Métaheuristique

Algorithme génétique

les premiers travaux sur les algorithmes génétiques remontent aux années 60.Il s'agissait, à l'époque, de recherches menées par les biologistes pour simuler les mécanismes naturels. Il faut en fait attendre la fin des années 60 pour que Holland propose une véritable formalisation de l'application de la théorie de l'évolution au domaine de l'Optimisation. Cette métaheuristique a rapidement suscité un grand intérêt en Intelligence Artificielle et en Recherche Opérationnelle, en partie en raison de la très forte analogie avec les principes d'évolution des espèces.

La reproduction sexuée met en jeu deux mécanismes fondamentaux : le premier est le croisement génétique. Il implique la combinaison du patrimoine génétique des deux parents pour former le patrimoine de l'enfant. Il permet ainsi de former des individus distincts, bien que possédant des traits propres à chaque parent. Le second mécanisme est la mutation. Elle intervient durant la phase de croisement et se traduit par la modification spontanée de quelques gènes. Ces deux mécanismes sont complémentaires : le croisement est le support naturel de la reproduction. Il assure à ce titre la pérennité démographique de l'espèce. La mutation permet l'évolution globale de l'espèce en introduisant de nouvelles caractéristiques d'une génération à l'autre. Si l'on suppose que l'environnement n'est pas fermé, nous pouvons ajouter un troisième mécanisme, l'immigration, destiné à simuler l'arrivée d'individus étrangé au sein de la population.

- Génération de la population initiale

La population initiale est obtenue par les résultats des heuristiques $H1, H3, \ldots, H5$ et les meilleurs solutions de l'heuristique $H2$, la taille de la population varie selon le nombre de tâches, et la taille maximal de la population est égale à 100.

- Codage des donnés

Dans la littérature diverses méthodes de codage sont utilisées pour coder les problèmes d'ordonnancements, le codage basé sur les tâches (job-based encoding method), le codage basé sur les machines (machine-based encoding method) et la méthode de codage basée

sur les opérations (operation-based encoding method), cette dernière méthode est souvent la plus employée pour les problèmes de flow shop. Cette étude adopte la méthode de codage basée sur les opérations. L'exemple ci-dessous donne la façon de coder trois tâches $T = \{T_1, T_2, T_3\}$ en utilisons la méthode de codage basée sur les opérations telle que chaque nombre i du vecteur représente une opération de la tache T_i et comme chaque tache se traite sur M_1 puis sur M_2 et elle se traite une deuxième fois sur M_1 donc chaque nombre i apparait trois fois dans le vecteur voir figure ci-dessous :

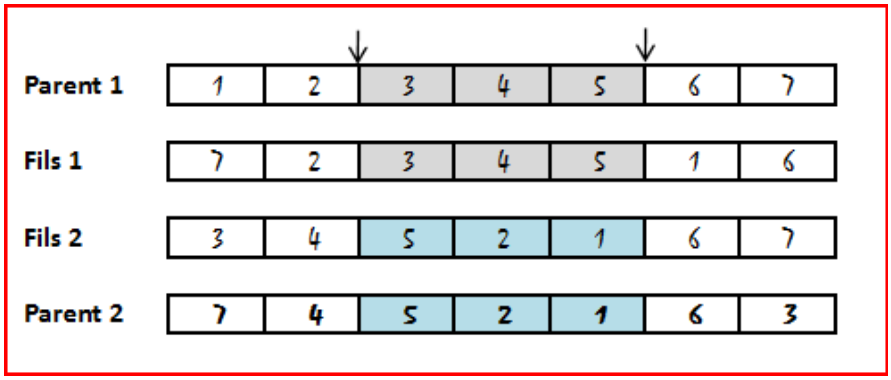
Ordonnancement de trois tâches selon le codage basé sur les opérations.

- **Le croisement (crossover)** combine la moitié des chromosomes des deux parents pour générer le chromosome de l'individu fils. Dans le mécanisme naturel de reproduction, le reste du patrimoine génétique des parents est perdu. Dans le monde des algorithmes génétiques, il est récupéré pour former un second descendant. Ainsi, le crossover est un opérateur produisant deux nouveaux individus à partir des deux individus parents. Traditionnellement, le crossover ne possède qu'un point de coupe, dont l'emplacement est déterminé de manière aléatoire. On peut néanmoins étendre le principe en utilisant deux points de coupe (figure ci-dessous) et même autant de points de coupe que d'allèles dans le cas du crossover uniforme. Dans notre algorithme génétique on a choisit le croisement avec deux point de coupe avec une probabilité de 0.8. Donc pour avoir par exemple le Fils1 on suit les étapes suivantes :

- On choisit deux positions aléatoires sur le premier parent et on reproduit le premier parent sur le premier fils entre ces deux positions.

- On parcoure ensuite le premier parent entre les deux positions de gauche a droite et on coche les casses correspondantes dans le deuxième parent.

- Enfin on lit dans le parent 2 de gauche à droite les casses non coché et on remplit les casses vides du premier fils. Voir figure ci-dessous :



Crossover deux points de coupe

- La mutation produit un individu fils en altérant quelques allèles de l'individu père. Bien entendu, l'importance de la mutation est directement liée à la quantité d'allèles touchés. À l'origine, la mutation était purement stochastique, les loci étaient déterminés aléatoirement. À l'heure actuelle, on s'oriente de plus en plus vers des formes plus évoluées de mutations, mettant souvent en œuvre des heuristiques d'où l'opérateur de mutation utilisé dans notre cas est l'heuristique $H1$ (recherche locale) avec une probabilité de 0.1.

- Critère d'arrêt

L'algorithme génétique continu à s'exécuter jusqu'a ce qu'un critère d'arrêt est satisfit. Notre algorithme s'arrête après un nombre fixe de générations.

Le schéma d'un algorithme génétique est présenté dans la figure ci-dessous.

(0) Phase d'initialisation
générer la population initiale P_0

(1) Phase d'évolution de la population

Répéter

$P' \leftarrow P_i$

choisir les solutions à recombiner

effectuer les recombinaisons

insérer les solutions résultantes dans P'

choisir les solutions à faire muter

effectuer les mutations

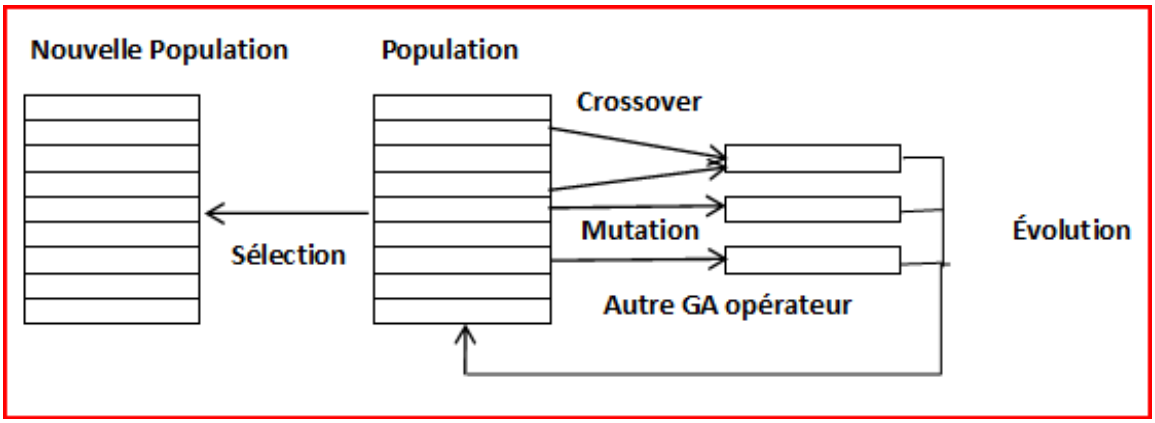
insérer les solutions résultantes dans P'

appliquer la sélection $P \rightarrow P^{i+1}$

tant que critère d'arrêt non satisfait.

retourner la solution record

Schéma classique des algorithmes génétiques

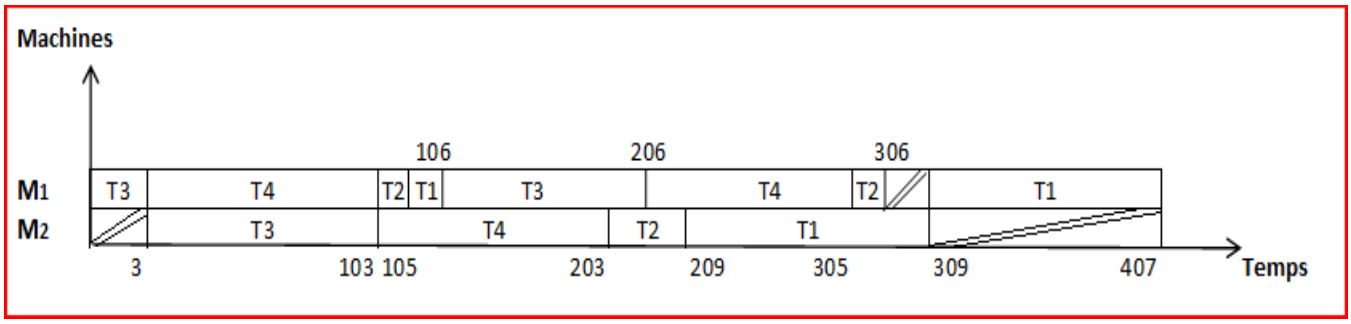


Exemple

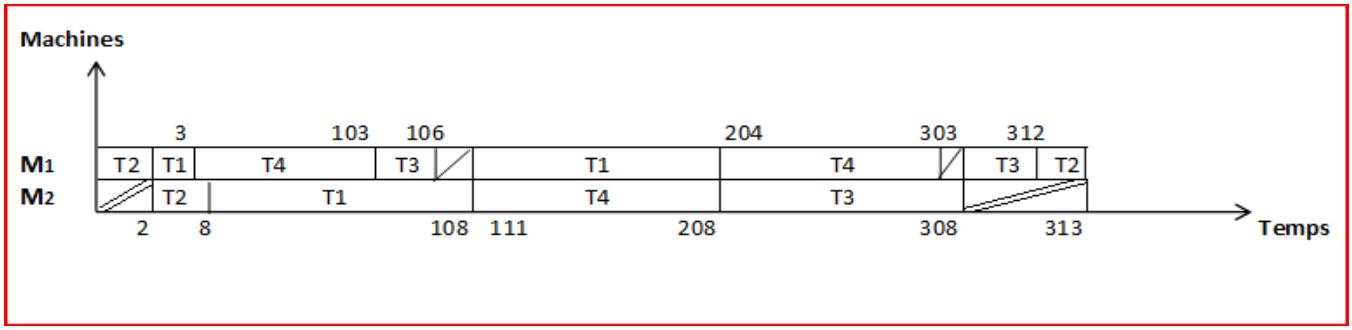
Dans l'exemple qui suit, on applique les 5 heuristiques précédentes à l'exemple de Inna G. Drobouchevitch et al.[17]. Si on prend $M = 100$, on voit que $C_{max}(S_H) = 4M + 1 = 401$. Les temps de traitement sont données dans le tableau suivant :

	$T1$	$T2$	$T3$	$T4$
$M1$	1	2	3	100
$M2$	100	6	100	100
$M1$	98	1	4	99

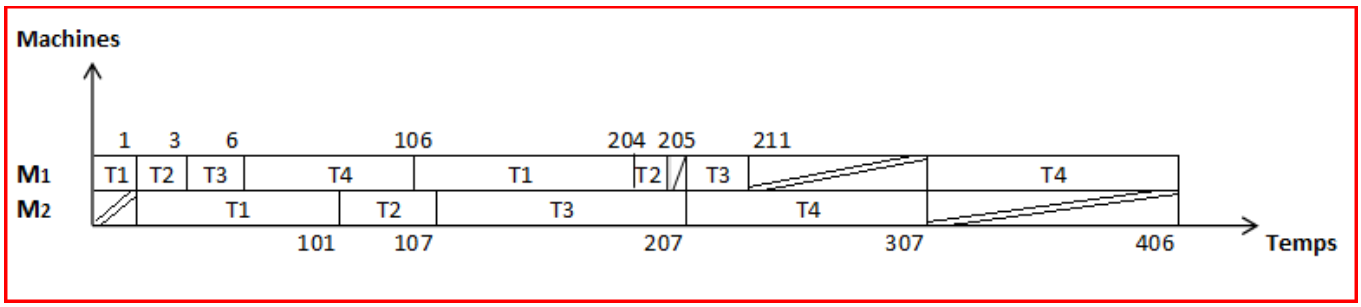
(Exemple)



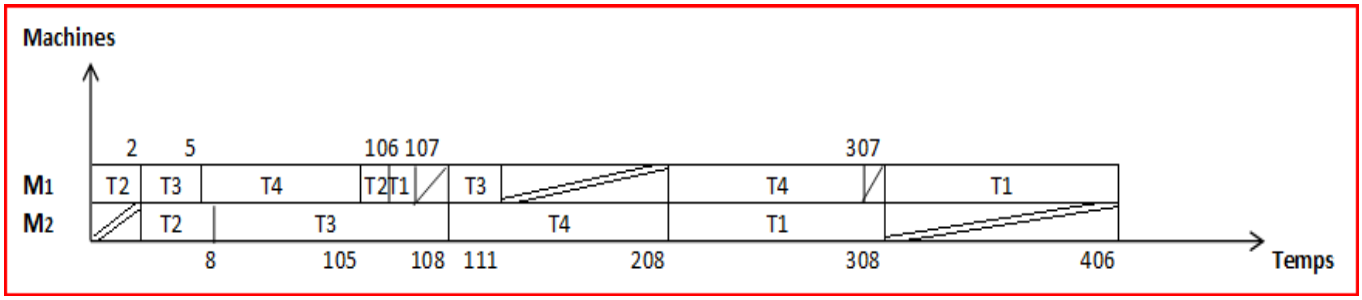
Application de l'heuristique H1



Application de l'heuristique H2



Application de l'heuristique H3 et H5



Application de l'heuristique H4

On remarque que pour cet exemple, l'heuristique H2 donne la meilleure solution. Cette dernière permet d'exploiter des zones du domaine des solutions réalisables non visités par les autres heuristiques, d'où l'utilité de l'utiliser comme une des solutions pour la géneration de la population initiale.

Chapitre 4

Expérimentations et interprétation des résultats

4.1 Introduction

Dans ce chapitre, on commence par tester notre modèle mathématique, ce dernier est résolu par le solveur LINGO8 qui utilise la méthode branch and bound pour la résolution du programme mathématique. On a effectué aussi des tests sur les cinq heuristiques et la métaheuristique afin d'étudier leur performance. Pour la programmation, notre choix s'est porté sur " Microsoft visual C# " un langage récent. Il a été disponible en versions beta successives depuis l'année 2000 avant d'être officiellement disponible en février 2002 en même temps que la plate-forme. NET 1.0 de Microsoft à laquelle il est lié. Vu son importance, " Microsoft visual C# " est considéré comme un environnement de programmation puissant qui offre la possibilité de manipuler des applications de différents domaines. Les tests sont effectués sur un Intel Corp 2 Duo 2 GHz et 3 Go de RAM.

4.2 Test du modèle

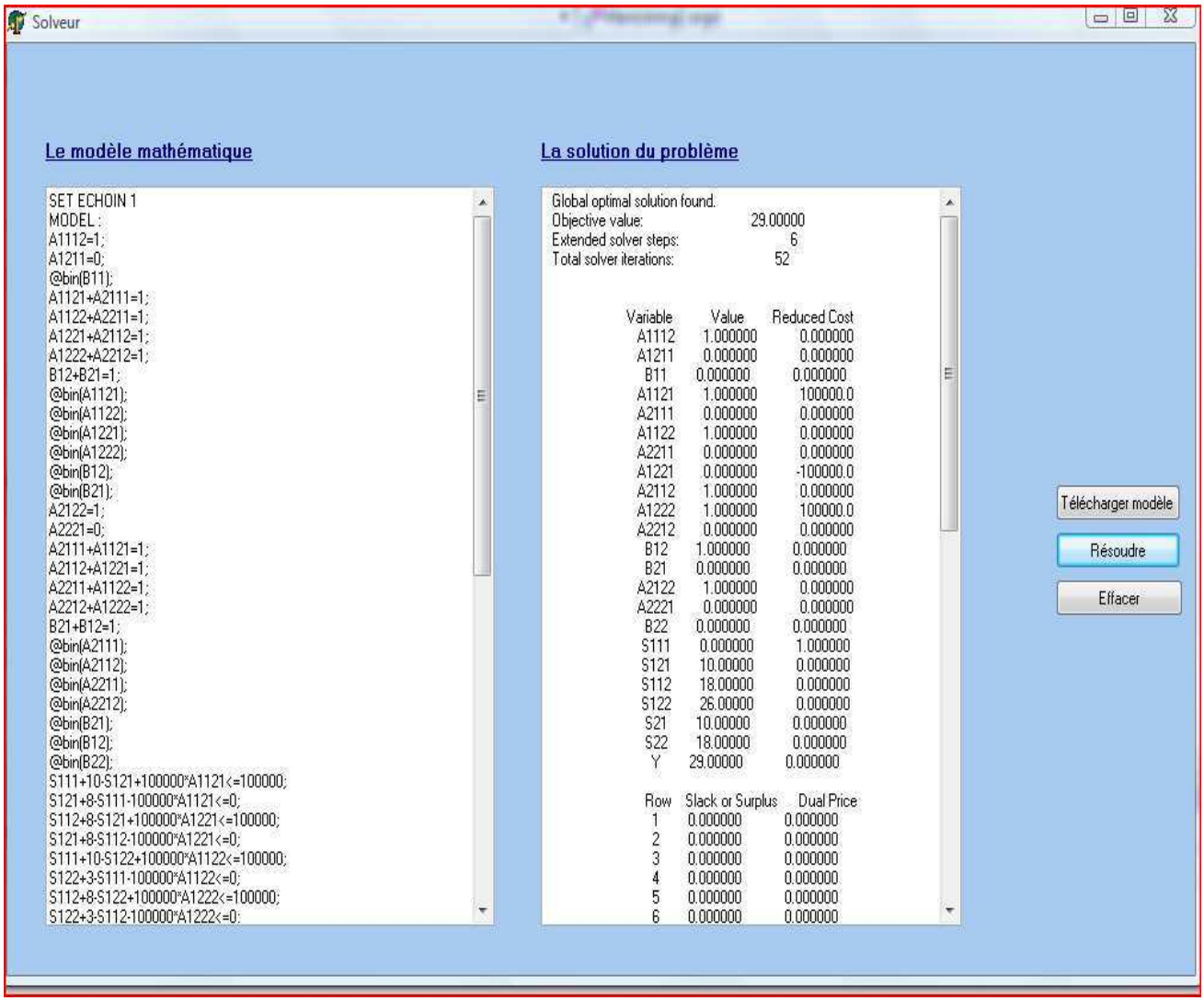
Le déroulement de notre programme est comme suit

1. Générer le modèle à l'aide d'une application programmée sous " Microsoft Visual C# ", cette étape permet de créer un fichier texte nommer Model qui contient le modèle mathématique de notre problème.

2. Télécharger les données en cliquant sur le bouton télécharger modèle de l'application

programmée sous Delphi7, ceci permit de copier le modèle dans un composant Richedit1.

3. Cliquer sur le bouton résoudre pour faire la liaison entre le modèle mathématique généré et le solveur LINGO, la solution est affichée sur le Richedit2 .Voir figure ci-dessous.



Nous constatons que la durée d'exécution augmente d'une manière exponentielle en augmentant le nombre de tâches "n" comme le montre le tableau suivant :

Nombres de tâches	durée d'exécution
2	0 *seconde*
3	1 *secondes*
4	5 *secondes*
5	5 *minutes* 58 *secondes*
6	6 *h* 6 *minutes* 58 *secondes*
7	$\geq$ 24 *heures*

4.3 Génération des données

Les paramêtres du problème sont :

– les temps de traitement sur les deux machines p_{1i1}, p_{1i2} et p_{2i} .

Ces paramêtres sont générés aléatoirement suivant une loi uniforme. Plusieurs instances sont également générées et pour chaque instance le nombre de tâches n prend ses valeurs dans l'ensemble $\{10, 20, 50, 100, 500\}$.

4.4 Déroulement des tests

On refait les mêmes testes donnés dans l'article de M. Y. Wang, et al.[3], la première étape consiste à fixer le nombre de tâches n. On génére, plusieurs instances du même problème (200 instances pour chaque cas), selon la loi uniforme et à chaque cas on calcule le pourcentage où la solution trouvée par l'heuristique H est meilleure par rapport aux autres, on compare aussi nos heuristiques au heuristiques de M. Y. Wang. Si la borne inferieure est atteinte par une de ces heuristiques on arréte sinon on lance l'algorithme génétique afin d'améliorer d'avantage la solution si c'est possible et on comapre la valeur donné par ce dernier avec les autre heuristiques, on calcule aussi le pourcentage ou la valeur du Cmax est égale à la borne inférieure ainsi que la durée d'exécution moyenne de chaque heuristique (en milliseconde).

Nous définissons donc les paramêtres à calculer :

– $pCmax$: le pourcentage pour lequel l'heuristique H fournit une solution meilleurs que les autres heuristiques.

– opt : le pourcentage pour lequel la solution obtenue par l'heuristique H coincide avec la borne inférieure "LB".

– $Rapmoyen1$: moyenne des rapports de performance .

– $Rapmax$: le maximum des rapports de performance.

– $Durée - moy$: la durée moyenne d'exécution.

4.5 Tests selon la loi uniforme

Nous supposons que les durées opératoires des tâches sur les deux machines suivent une loi uniforme les résultats obtenus sont représentés dans le tableau suivant :

Experimentation 1 $p_{1i1} \in [10, 150]$, $p_{2i} \in [20, 200]$, $p_{1i2} \in [10, 100]$.

	$H1$	$H2$	$H3$	$H4$	$H5$	AG
$n = 10 \quad Pcmax$	11%	53.5%	71.5%	32%	17%	100%
Opt	10%	52.5%	69%	23.5%	9%	81.5%
$Durée - Moy$	11	16	7	12	11	79
$Rap - Moy$	1, 1695	1, 029321	1, 016657	1, 027752	1, 058138	1, 003812
$Rap - Max$	1, 607752	1, 218876	1, 191555	1, 153917	1, 221053	1, 051338
$n = 20 \quad Pcmax$	8%	46%	85%	21.5%	14%	100%
Opt	8%	46%	84%	18.5%	7%	87.5%
$Durée - Moy$	12	25	8	19	11	121
$Rap - Moy$	1, 151718	1, 02672	1, 00663	1, 021471	1, 033594	1, 002217
$Rap - Max$	1, 495671	1, 159726	1, 098479	1, 173099	1, 112015	1, 041277
$n = 30 \quad Pcmax$	2%	34%	92.5%	26.5%	8%	100%
Opt	2%	33%	92%	22.5%	4.5%	93%
$Durée - Moy$	15	44	8	39	18	166
$Rap - Moy$	1, 139024	1, 027538	1, 002633	1, 012571	1, 024631	1, 000484
$Rap - Max$	1, 388533	1, 185216	1, 083916	1, 143709	1, 06872	1, 017519
$n = 50 \quad Pcmax$	3%	16.5%	96.5%	14%	3%	100%
Opt	2, 5%	16.5%	96.5%	12.5%	2.5%	97%
$Durée - Moy$	20	146	9	139	27	462
$Rap - Moy$	1, 114263	1, 033458	1, 001258	1, 010055	1, 018209	1, 000373
$Rap - Max$	1, 323375	1, 150498	1, 050196	1, 09657	1, 044015	1, 024802
$n = 100 \quad Pcmax$	0%	33%	99%	15%	0%	100%
Opt	0%	33%	99%	15%	0%	99.5%
$Durée - Moy$	31	502	11	597	147	1435
$Rap - Moy$	1, 099179	1, 014629	1, 000279	1, 004877	1, 01088	1, 000007
$Rap - Max$	1, 296464	1, 097709	1, 047161	1, 061075	1, 028283	1, 001605

Experimentation 2 $p_{1i1} \in [10, 100]\,, p_{2i} \in [20, 200]\,, p_{1i2} \in [10, 100]\,.$

	$H1$	$H2$	$H3$	$H4$	$H5$	GA
$n = 20 \quad Pcmax$	0.5%	18%	32%	28.5%	31%	100%
Opt	0.5%	15.5%	28%	7%	13%	42%
$Durée - Moy$	13	24	9	20	12	121
$Rap - Moy$	1, 204154	1, 052676	1, 045991	1, 032447	1, 026312	1, 009058
$Rap - Max$	1, 552391	1, 153587	1, 205389	1, 171254	1, 096222	1, 046882
$n = 50 \quad Pcmax$	0%	1%	18%	22%	46, 5%	100%
Opt	0%	1%	16%	3, 5%	11%	28%
$Durée - Moy$	19	165	10	139	27	462
$Rap - Moy$	1, 187189	1, 101357	1, 040162	1, 022211	1, 011972	1, 006239
$Rap - Max$	1, 372923	1, 166465	1, 120914	1, 107085	1, 044295	1, 033056
$n = 200 \quad Pcmax$	0%	0%	12%	7.5%	63%	100%
Opt	0%	0%	10.5%	0%	5%	17.5%
$Durée - Moy$	53	4190	12	5979	1250	14 s
$Rap - Moy$	1, 161264	1, 092431	1, 03866	1, 017063	1, 003182	1, 00209
$Rap - Max$	1, 271197	1, 131589	1, 076688	1, 066234	1, 013947	1, 008112
$n = 500 \quad Pcmax$	0%	0%	0.5%	16.5%	76.5%	100%
Opt	0%	0%	0.5%	2%	3.5%	6.5%
$Durée - Moy$	259	26 s	18	2 m 14 s	1435	3 m 21 s
$Rap - Moy$	1, 158346	1, 115302	1, 043151	1, 016428	1, 001387	1, 0011
$Rap - Max$	1, 210435	1, 133984	1, 076591	1, 064585	1, 005337	1, 00375

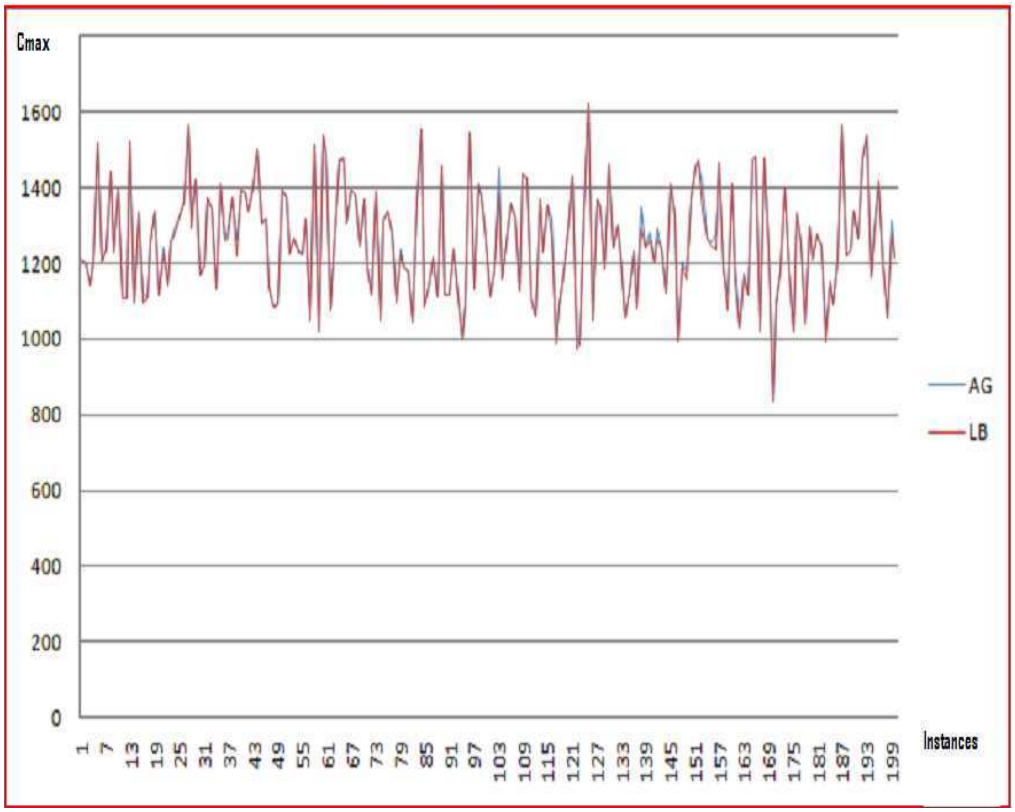
Experimentation 3 $p_{1i1} \in [10, 100]\,, p_{2i} \in [30, 300]\,, p_{1i2} \in [0, 150]\,.$

	$H1$	$H2$	$H3$	$H4$	$H5$	AG
$n = 20 \quad Pcmax$	0%	4%	4.5%	31.5%	34.5%	100%
Opt	0%	$1,5\%$	3%	4%	$6,5\%$	$14,5\%$
$Durée - Moy$	13	29	9	20	11	123
$Rap - Moy$	$1,191068$	$1,052892$	$1,070302$	$1,032661$	$1,030295$	$1,013306$
$Rap - Max$	$1,4616$	$1,151387$	$1,191015$	$1,193511$	$1,125973$	$1,056576$
$n = 50 \quad Pcmax$	0%	0%	0%	31.5%	49%	100%
Opt	0%	0%	0%	0%	3%	4%
$Durée - Moy$	20	146	10	140	26	469
$Rap - Moy$	$1,149851$	$1,07224$	$1,061634$	$1,016821$	$1,013124$	$1,008292$
$Rap - Max$	$1,31368$	$1,14823$	$1,140217$	$1,070762$	$1,054443$	$1,028771$
$n = 200 \quad Pcmax$	0%	0%	0%	7%	86.5%	100%
Opt	0%	0%	0%	0%	1%	2%
$Durée - Moy$	53	4190	12	5979	1250	14 s
$Rap - Moy$	$1,108823$	$1,053122$	$1,050648$	$1,015307$	$1,003232$	$1,002914$
$Rap - Max$	$1,201706$	$1,102219$	$1,087307$	$1,055521$	$1,011587$	$1,011587$
$n = 500 \quad Pcmax$	0%	0%	0%	0%	98%	100%
Opt	0%	0%	0%	0%	1%	1%
$Durée - Moy$	259	25 s	18	2 m 20 s	1435	3 m 19 s
$Rap - Moy$	$1,10133$	$1,0639$	$1,049314$	$1,017059$	$1,001313$	$1,001307$
$Rap - Max$	$1,162498$	$1,088324$	$1,06858$	$1,04257$	$1,004794$	$1,004794$

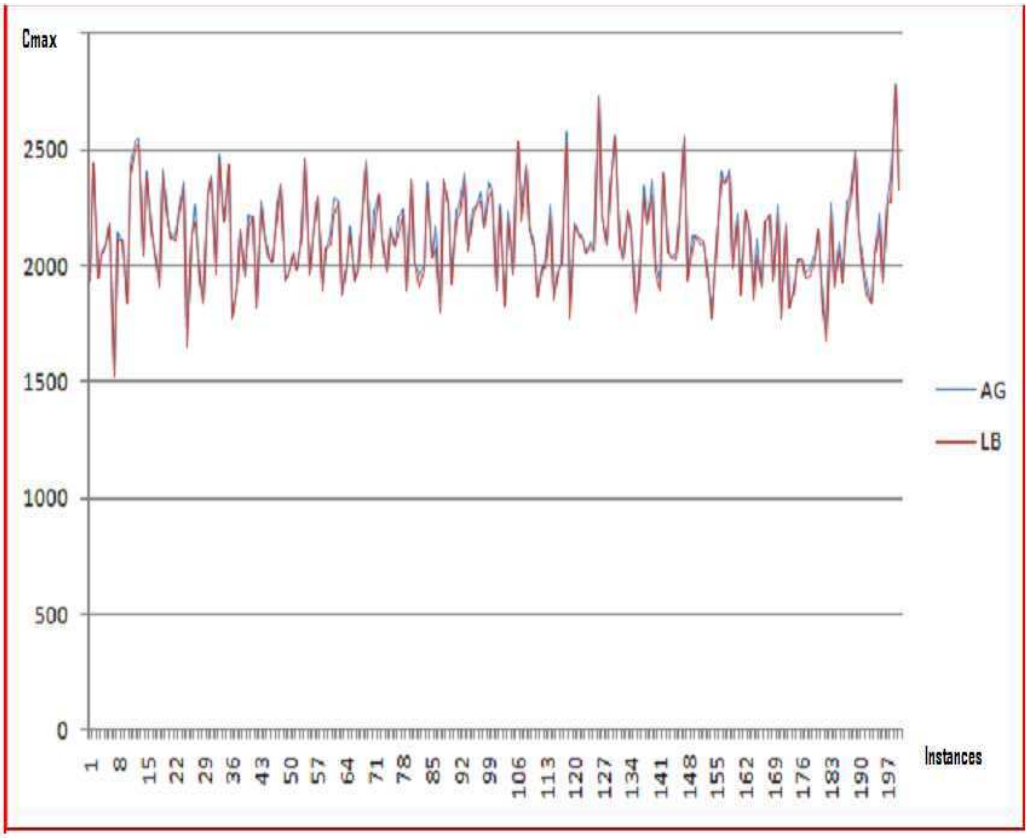
Experimentation4 $p_{1i1} \in [10, 100]\,, p_{2i} \in [40, 400]\,, p_{1i2} \in [10, 150]\,.$

	$H1$	$H2$	$H3$	$H4$	$H5$	AG
$n = 20 \quad Pcmax$	0%	3%	6.5%	33%	32%	100%
Opt	0%	0%	1.5%	6.5%	4.5%	15%
$Durée - Moy$	14	35	7	19	11	125
$Rap - Moy$	1, 129558	1, 031489	1, 051317	1, 020217	1, 0223	1, 008879
$Rap - Max$	1, 327	1, 089815	1, 19552	1, 082443	1, 099163	1, 041152
$n = 50 \quad Pcmax$	0%	0.5%	0.5%	25%	41%	100%
Opt	0%	0%	0%	1%	2%	3.5%
$Durée - Moy$	21	176	10	151	27	472
$Rap - Moy$	1, 086456	1, 030007	1, 03697	1, 012392	1, 008874	1, 004612
$Rap - Max$	1, 28	1, 076551	1, 104629	1, 109025	1, 028518	1, 020741
$n = 200 \quad Pcmax$	0%	0%	0%	21%	71%	100%
Opt	0%	0%	0%	0.5%	1%	2%
$Durée - Moy$	53	4199	12	5970	1269	14 s
$Rap - Moy$	1, 055826	1, 021027	1, 029843	1, 006501	1, 00216	1, 001582
$Rap - Max$	1, 112902	1, 047801	1, 053761	1, 024702	1, 009372	1, 005066
$n = 500 \quad Pcmax$	0%	0%	0%	7%	87%	100%
Opt	0%	0%	0%	0%	0%	1%
$Durée - Moy$	261	27 s	19	2 m 24 s	1438	3 m 21 s
$Rap - Moy$	1, 046494	1, 023946	1, 028073	1, 005916	1, 000898	1, 000825
$Rap - Max$	1, 078315	1, 036756	1, 042421	1, 020293	1, 003491	1, 003491

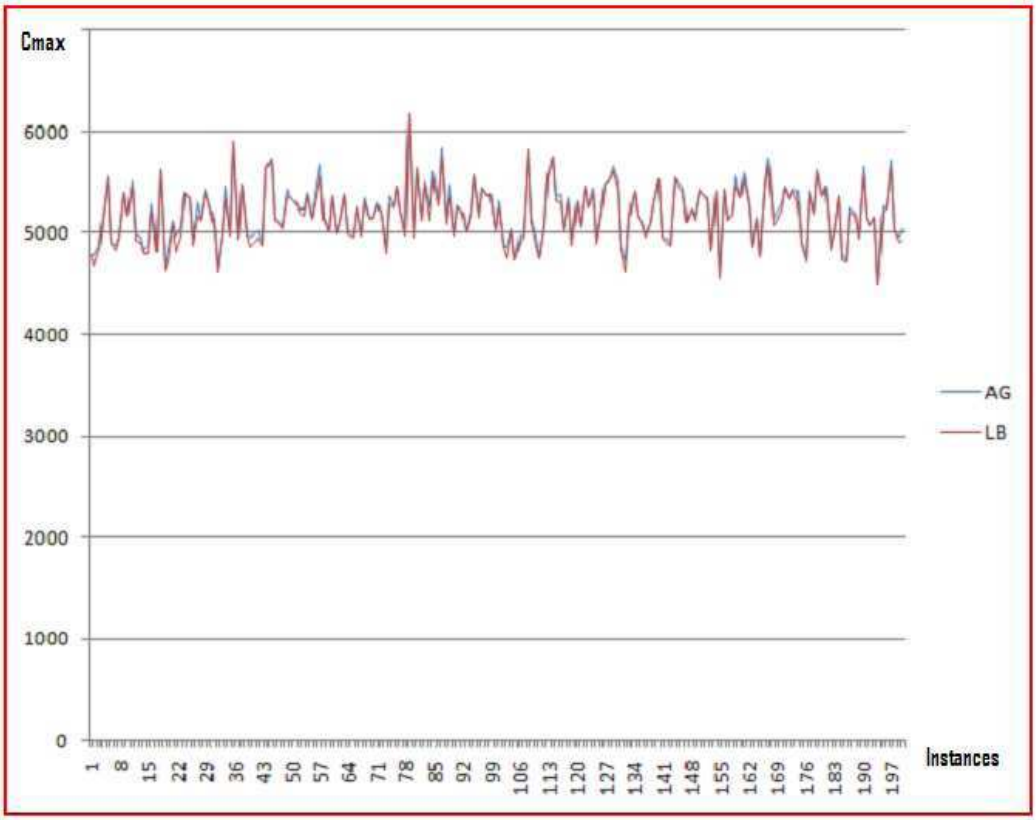
À la page suivante quelques graphes qui représentent les valeurs obtenues par l'algorithme Génétique et les valeurs de la borne inférieure



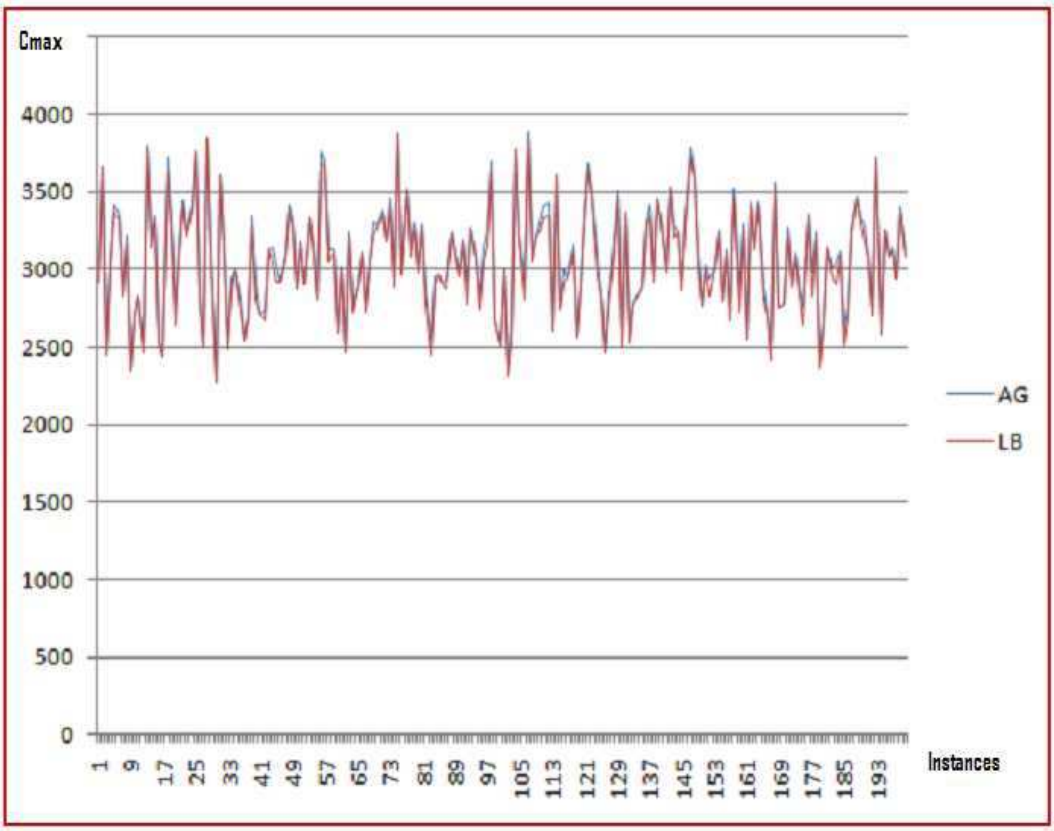
Experiment 1 pour n=10 tâches



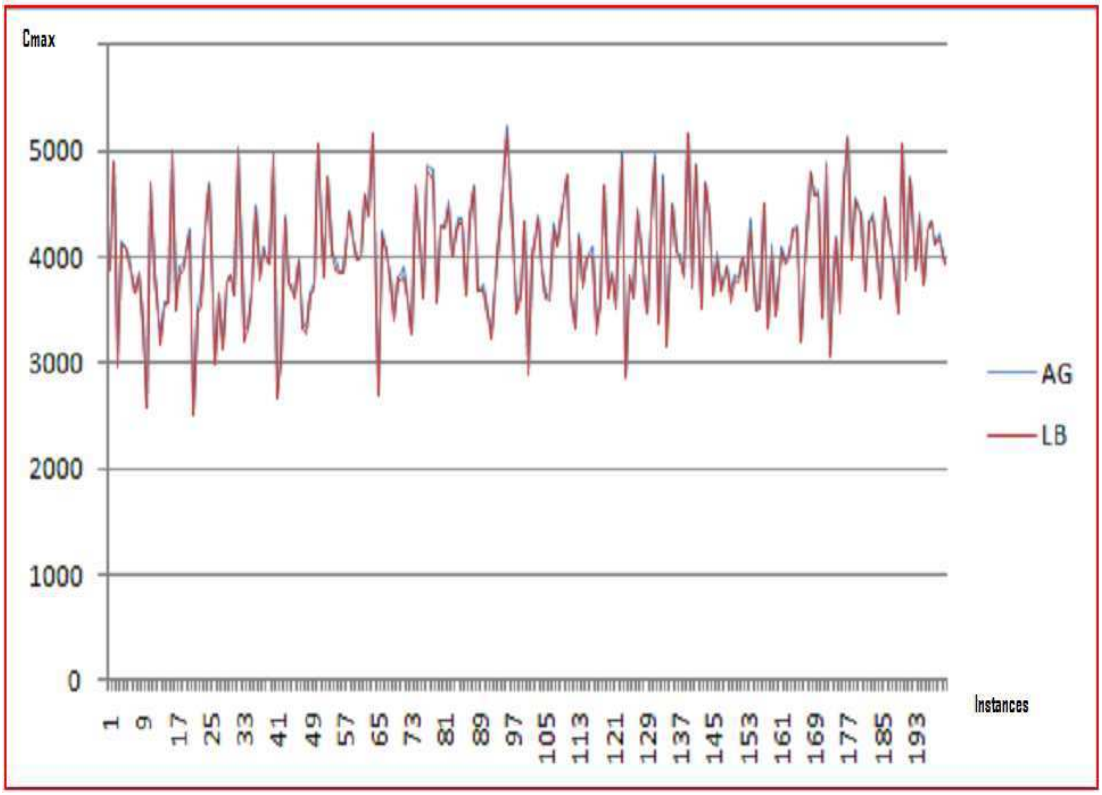
Experiment 2 pour n=20 tâches



Experiment 2 pour n=50 tâches

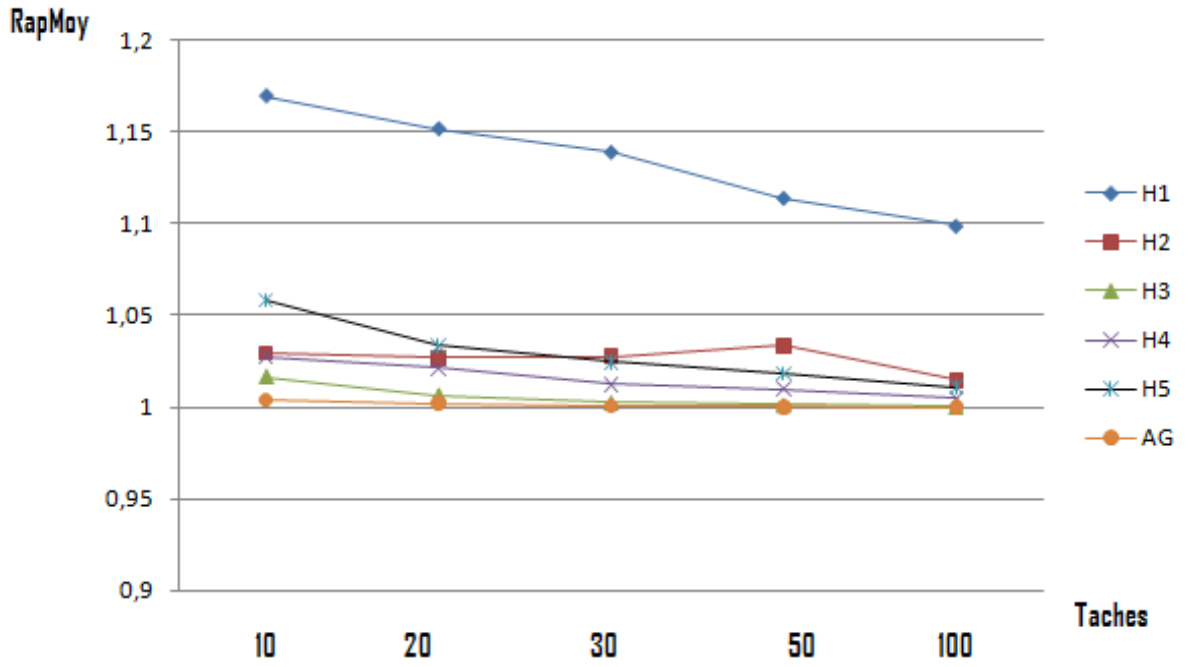


Experiment 3 pour n=20 tâches

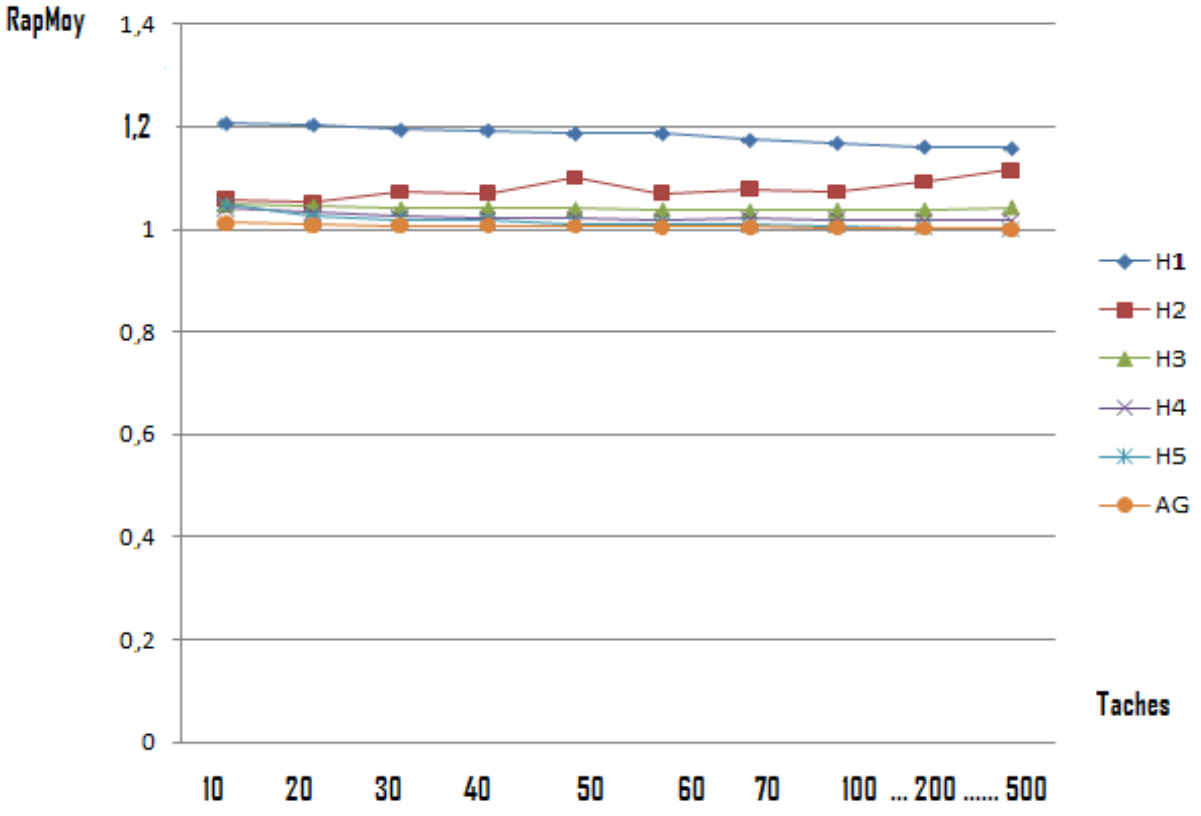


Experiment 4 pour n=20 tâches

Pour les différentes heuristiques, nous avons établi le rapport de performance moyen en appliquant les heuristiques sur des instances construites aléatoirement selon une loi uniforme. Les résultats pour l'expérimentation1 et 2 sont illustrés dans les graphes suivants :

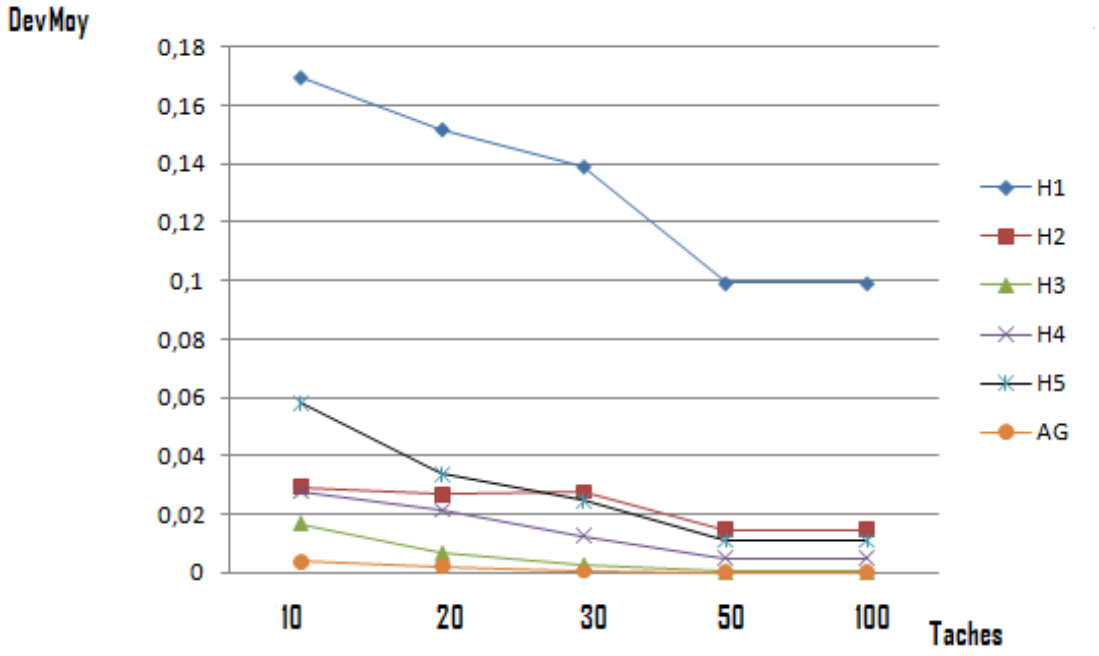


Rapport moyen pour l'expérimentation 1

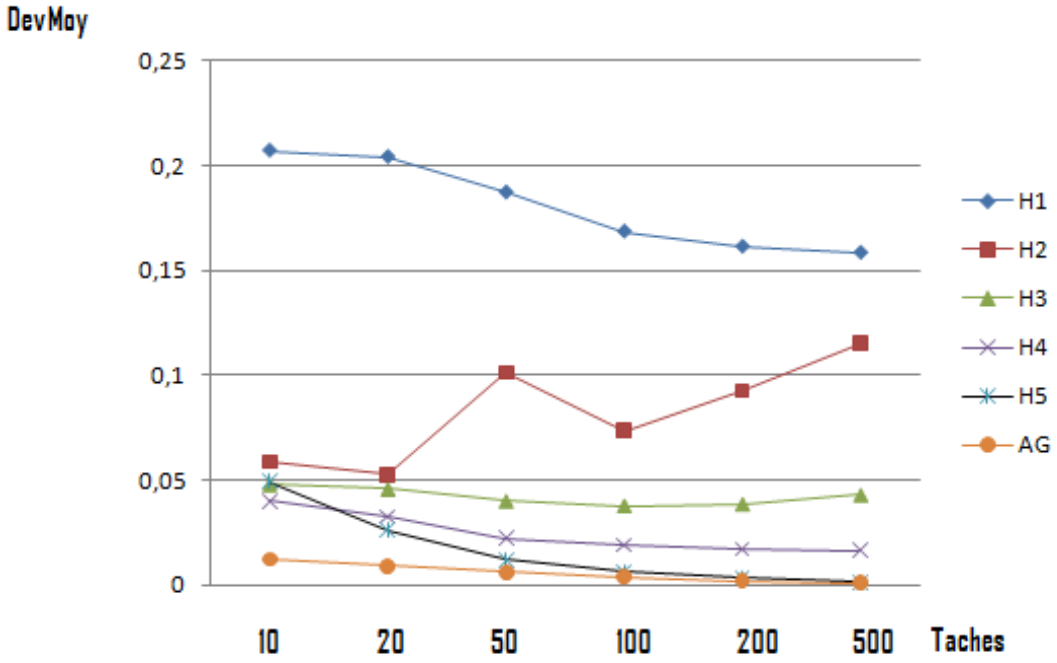


Rapport moyen pour l'expérimentation 2

En ce qui concerne les déviations moyennes, voici quelques graphes obtenus :



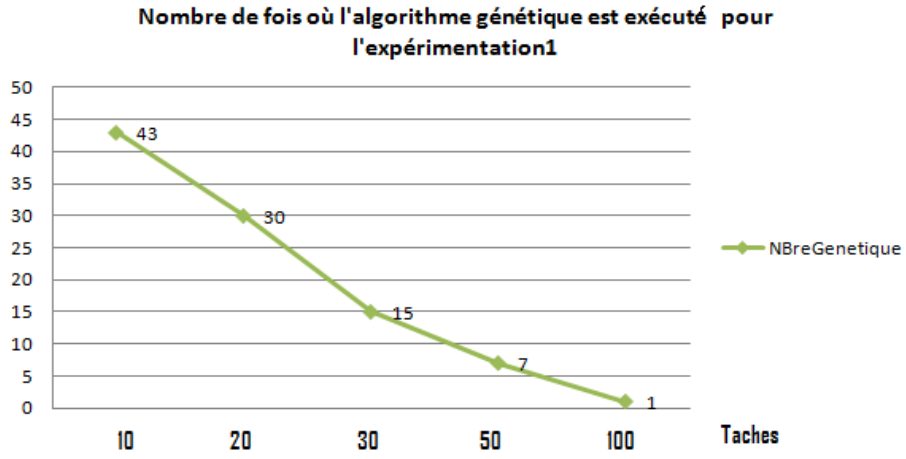
Déviation moyenne pour l'expérimentation 1

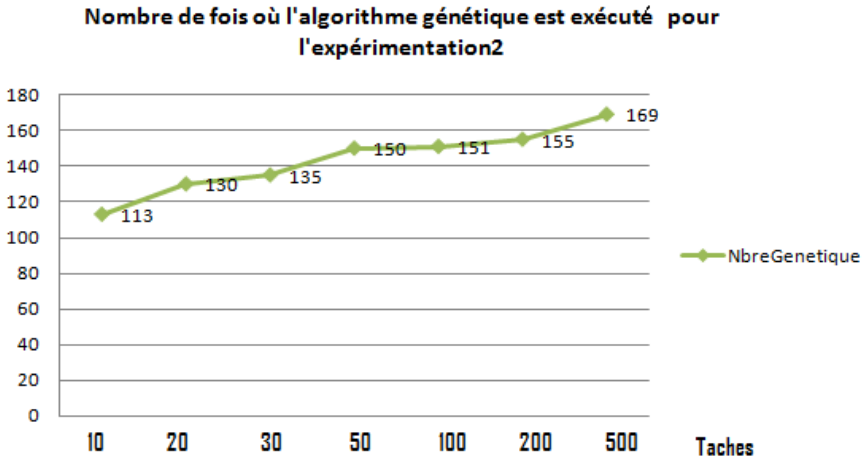


Déviation moyenne pour l'expérimentation 2

4.6 Conclusion

Après la réalisation de plusieurs tests, nous avons constaté que pour l'expérimentation1 le nombre de fois où on appelle l'algorithme génétique diminue en augmentant le nombre de tâches.Par contre, pour les expérimentations 2, 3 et 4 le nombre de fois où on fait appel à l'algorithme génétique augment en augmentant le nombre de tâches (voir figure ci-dessous), ce qui signifie que pour l'experimentation 1 les solutions obtenues par les heuristiques est particulièrement les solution de l'heuristique H3 coïncides avec les bornes inférieures et ce pourcentages augmente en augmentant le nombre de tâches (on a opt=69 pour n=10 tâches et opt=99 pour n=100 tâches), ce qui nous laisse dire que l'heuristique H3 est la meilleure pour l'experimentation1 mais ceci n'est pas vraie pour les experimentations 2, 3 et 4 car les solutions des heuristiques coïncide rarement avec les bornes inferieures et ce pourcentages diminue en augmentant le nombres de tâches ; par contre l'heuristique H5 est jugée la meilleure, puisqu'elle donne toujours les meilleures résultats par rapport aux autres heuristiques. L'heuristique H1 est la plus mauvaise, ceci est attendu puisque son principe est basé sur une amélioration progressive de la solution, et donc elle reste bloquée dans un minimum local dès qu'elle le rencontre. Nous remarquons aussi que l'algorithme génétique a donné de très bons résultats qui sont très proches des solutions optimales (voir les figures ci-dessus) mais il nécessite un effort de calcul considérable, ceci est du aux solutions de départ notamment de l'heuristique H4 MLBB qui est en $O(n^3)$ d'après [6] ; une des solutions pour remédier à ce problème et de remplacer l'heuristique H4 (MLBB) par l'heuristique H3 de M. Y. Wang, et al.[28] et remplacer aussi l'heuristique H1 (Recherche local) par l'heuristique de Inna G. Drobouchevitch et al.[17] ; ceci permit de gagner en complexité de l'algorithme génétique et en qualité de la solution trouvée .


Nombre de fois où l'algorithme génétique est exécuté pour l'expérimentation1
50
45
40
35
30
25
20
15
10
5
0
43
30
15
7
1
10
20
30
50
100
Taches
NBreGenetique


Nombre de fois où l'algorithme génétique est exécuté pour l'expérimentation2
180
160
140
120
100
80
60
40
20
0
113
130
135
150
151
155
169
10
20
30
50
100
200
500
Taches
NbreGenetique

Conclusion générale

La recherche en ordonnancement a beaucoup approfondie les 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 le problème d'ordonnancement de tâches non préemptives, de type Flow shop avec recirculation de tâches sur la première machine dont le but est de minimiser le makespan. Ce problème étant NP-difficile en général.

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

Nous avons déduit quelques cas particuliers où le problème est polynomial, et étant donné que notre problème est en général NP-difficile, nous avons proposé cinq heuristiques basées sur la règle de Johnson et sur la recherche aléatoire dans le voisinage des solutions, On a proposé aussi un algorithme génétique, ce dernier s'adapte bien aux problèmes d'ordonnancement de type flow shop et il donne des solutions de bonne qualité et il sera plus efficace si on remplace l'heuristique H4 (MLBB) par l'heuristique H3 de M. Y. Wang, et al.[28] et on remplace aussi l'heuristique H1 (Recherche local) par l'heuristique de Inna G. Drobouchevitch et al.[17]. L'heuristique H2 fournit la plupart des solutions de la population initiale de l'algorithme génétique, cette heuristique permet d'exploiter des zones du domaine des solutions réalisables non visités par les autres heuristiques (voir l'exemple d'application ci-dessus), d'où l'utilité de l'utiliser pour générer les individus de la population initiale.

Des expérimentations numériques ont été effectuées en réalisant une application informatique implémentée sous " Microsoft visual C# ". Les durées opératoires des tâches sont générées aléatoirement suivant une loi uniforme.

Les perspectives de ce travail sont multiples. Nous envisageons quelques unes :

– Développer d'autres métaheuristiques ou d'autres méthodes exactes.

– Analyser d'autres modèles en considérant d'autres contraintes et d'autres critères.

– Appliquer les méthodes développées sur un problème industriel.

Bibliographie

[1] A. Agnetis. Scheduling no-wait robotic cells with two and three machines. European Journal of Operational Research, 123 :303–314, 2000.

[2] M. Benoit and JC. Billaut. 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.

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

[4] P. Brucker, S. Knust, "Complexity results for singlemachine problems with positive finish-start time-lags",Computing, 1999, 63 : 299-316.

[5] Caixia Jing,Xingsan Qian and Guochun Tang Two-machine Flow Shop Scheduling with Re-entrance International Conference on Information Management, Innovation Management and Industrial Engineering 2008.

[6] Caixia Jing, Guochun Tang, Xingsan Qian Heuristic algorithms for two machine re-entrant flow shop Theoretical Computer Science 400 (2008) 137–143

[7] J. S. Chen, J. C. H. Pan, C. K. Wu, "Hybrid tabu search for re-entrant permutation flow-shop scheduling problem", Expert Systems with Applications, 2008,34(3) : 1924-1930.

[8] J. S. Chen, J. C. H. Pan, C. M. Lin, "A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem", Expert Systems with Applications, 2008,34(1) : 570-577.

[9] Christophe DUHAMEL Un Cadre Formel pour les Méthodes par Amélioration Itérative - Application à deux problémes d'Optimisation dans les Réseaux 2001.

[10] S. W. Choi, Y. D. Kim, "Minimizing makespan on an m-machine re-entrant flowshop", Computers and Operations Research, 2008, 35(5) : 1684-1696.

[11] S. W. Choi, Y. D. Kim, "Minimizing makespan on a two-machine re-entrant flowshop", Journal of the Operational Research Society, 2007, 58 : 972-981.

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

[13] L. Finta, Z. Liu, "Single machine scheduling subject to precedence delays", Discrete Appl. Math., 1996, 70 : 247-266.

[14] L. Graham, E. L. Lawler, "Optimization and approximation in deterministic scheduling : a survey",Ann. Discrete. Math., 1979, 5 : 287-326.

[15] J. Hao, P. Galinier, and M. Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. Revue d'Intelligence Artificielle, Hermes, Paris, 13(2) :283–324, 1999.

[16] N. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 3, 44 :510–525, 1996.

[17] Inna G. Drobouchevitch and Vitaly A. Strusevich A heuristic algorithm for two-machine re-entrant shop scheduling Annals of Operations Research 86(1999)417–439.

[18] S. M. Johnson. Optimal two and three stage production schedules with set up times included. Naval research Logistics Quaterly, 1 :61–68, 1954.

[19] J. K. Lenstra, A. H. G. Rinnooy Kan, "Complexity of machine scheduling problems", Annals of Discrete Mathematics, 1977, 1 : 343-362.

[20] Loic Yon Modéles et outils pour la conception stratégique de réseaux de transports publics PhD thesis 2005.

[21] Marc Sevaux "Métaheuristiques Stratégies pour l'optimisation de la production de biens et de services" Rapport de recherche.

[22] J. C.-H. Pan, J.-S. Chen, "Minimizing makespan in reentrant permutation flowshops", Journal of Operational Research Society, 2003, 57 : 642-653.

[23] M. Pinedo. Scheduling : theory, algorithms and systems. PhD thesis, Prentice Hall Series, New Jersey, 1995.

[24] H. Rock. The three-machine no-wait flow shop is np-complete. Journal of the Association for Computing Machinery 31, 2 :336–345, 1984.

[25] I. Saad. Conception d'un systéme d'aide à l'ordonnancement tenant compte des impératifs économiques. PhD thesis, Automatique et Informatique Industrielle, Ecole centrale de LILLE, 12 juillet 2007.

[26] M. Sakarovitch. Optimisation combinatoire : Méthodes mathématiques et algorithmiques. Edition Hermann, Paris, 1984.

[27] JF. D. Vargas-Villamil, D. E. Rivera, "A model predictive control approach for real-time optimization of reentrant manufacturing lines", Comput Ind, 2001,45 : 45-57.

[28] M. Y. Wang, S. P. Sethi, S. L. Van De Velde,"Minimizing makespan in a class of reentrant shops",Oper. Res, 1997, 45 : 702-712.

[29] D. L. Yang, W. H. Kuo, M. S. Chern, "Multi-family scheduling in a two-machine re-entrant flow shop with setups", European J. Oper. Res., 2008, 187(3) : 1160-1170.

[30] Meziani Nadajt " Ordonnancement sur machines specialisees " 2007.