7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris– France

« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coordination et Coopération ».

RÉSOLUTION EXACTE D'UN PROBLÈME D'ORDONNANCEMENT À UNE MACHINE SANS TEMPS D'ARRÊT

Fatma HERMÈS

Unité de recherche SOIE
Faculté des Sciences de Tunis
Université Tunis El Manar
Campus Universitaire - Tunis - 1060 Tunisie
fatma.hermes@fst.rnu.tn

Jacques CARLIER, Aziz MOUKRIM

Laboratoire Heudiasyc UMR CNRS 6599
Université de Technologie de Compiègne
Centre de Recherches de Royallieu BP 20529
60205 Compiègne cedex France
jacques.carlier@hds.utc.fr, aziz.moukrim@hds.utc.fr

Khaled GHÉDIRA

Unité de recherche SOIE
École Nationale des Sciences de l'Informatique
Université de la Manouba
Campus Universitaire - 2010 La Manouba Tunisie
khaled.ghedira@isg.rnu.tn

RESUMÉ : *Dans cet article, nous abordons un problème d'ordonnancement à une machine où la machine doit traiter les différents travaux qui lui sont attribués consécutivement sans temps d'arrêt (« no-idle »). Il s'agit donc de satisfaire la contrainte no-idle sachant que chaque travail a une date de début au plus tôt et n'est réellement terminé qu'après être resté une durée de latence dans le système, après la fin de son traitement sur la machine. Notre objectif est de minimiser la date de fin réelle de traitement de tous les travaux (f_{max}). Dans la littérature, ce problème a été étudié sans ajouter la contrainte no-idle. Si les travaux sont préemptifs, il est polynomial et il suffit d'appliquer la règle de Jackson pour trouver une solution optimale. Sinon, il est NP-complet mais il existe une méthode arborescente performante pour le résoudre. Nous proposons dans ce qui suit d'étudier ce problème en ajoutant la contrainte no-idle. Nous considérons dans un premier temps le cas où les travaux sont préemptifs puis nous étudions le cas où les travaux sont non préemptifs.*

MOTS-CLÉS : *problème à une machine, contrainte no-idle, durée de latence, préemption des travaux, résolution exacte.*

1. INTRODUCTION

La contrainte « no-idle » impose que les travaux soient traités sur chaque machine continuellement sans aucun temps-mort, et ceci depuis le début de traitement du premier travail affecté à la machine jusqu'à la fin de traitement du dernier travail. En l'absence de la contrainte « no-idle », les solutions obtenues minimisant un critère donné présentent souvent des intervalles de temps-morts de durées variables. Chacun de ces intervalles est un temps perdu pendant lequel la machine est, soit en marche mais attend le travail suivant, soit arrêtée et redémarrée lorsque le travail suivant est prêt pour être exécuté. Dans le premier cas il y a un gaspillage de temps et de capacité. Dans le deuxième cas, il y a une surcharge de mains d'œuvre. La présence de temps-morts empêche donc la réaffectation utile des moyens de production (opérateurs et machines), et entraîne un coût supplémentaire qui peut être considérable et qui a souvent été négligé.

Nous étudions dans cet article l'influence de l'ajout de la contrainte « no-idle » à un problème d'ordonnancement à une machine, où un travail se réduit à une seule tâche et la machine ne peut traiter qu'une seule tâche à la fois. Nous avons à traiter n travaux J_i ($i = 1 \ldots n$) où chaque travail J_i a une date de début au plus tôt r_i, une durée d'exécution sur la machine p_i et une durée de latence q_i. De ce fait, l'exécution d'un travail J_i ne peut commencer qu'à une date t_i supérieure ou égale à sa date de début au plus tôt r_i et nous avons ainsi : $t_i \geq r_i$. Il faut aussi que le travail J_i reste une durée q_i dans le système après la fin de son traitement sur la machine. Soit C_i la date de fin d'exécution du travail J_i sur la machine. Ce travail n'est réellement terminé qu'à la date $f_i = C_i + q_i$. Ainsi, le traitement de tous les travaux n'est réellement terminé

qu'à la date $f_{max} = \max_{i=1...n} (f_i) = \max_{i=1...n} (C_i + q_i)$. Nous nous intéressons dans cet article à minimiser le critère f_{max}. Pour des raisons de simplicité, nous allons parfois confondre J_i et i. Une solution réalisable σ pour ce problème est décrite par un ordre $i_1, i_2, \ldots, i_n$ d'exécution des travaux sur la machine tel qu'à chaque travail i est attribuée une date de début d'exécution t_i vérifiant les contraintes suivantes :

- le travail i_k a une date de début au plus tôt r_{i_k} et de ce fait nous avons $t_{i_k} \geq r_{i_k} \quad \forall i_k = 1 \ldots n$;
- les travaux doivent être exécutés consécutivement sans aucun intervalle de temps-mort et par la suite nous avons $t_{i_{k+1}} = t_{i_k} + p_{i_k} = C_{i_k} \quad \forall k = 1 \ldots n - 1$.

Dans la section 2 nous décrivons brièvement l'état de l'art des recherches s'intéressant à ce problème ainsi qu'à l'ajout de la contrainte no-idle. Dans la section 3, nous présentons quelques concepts de base soulevant la représentation d'une solution, l'algorithme de Jackson et la procédure par séparation et évaluation proposée par Carlier pour résoudre le problème sans preemption et sans la contrainte no-idle. Dans la section 4, nous étudions ce problème en ajoutant la contrainte no-idle dans le cas où les travaux sont préemptifs. Dans la section 5, nous étudions ce problème en ajoutant la contrainte no-idle dans le cas où les travaux sont non préemptifs. Les résultats numériques sont présentés dans la section 6.

2. ÉTAT DE L'ART

La notation $\alpha/ \beta/ \gamma$ permet de référencer rapidement un problème d'ordonnancement. Le champ α est réservé à la typologie des machines du problème référencé. Le champ β indique l'ensemble des contraintes à vérifier par ce problème et le champ γ indique le (ou les) critère(s) à optimiser.

Le problème où chaque travail J_i a une date de début au plus tôt r_i, une durée de latence q_i et tel que le critère à optimiser est f_{max} est noté $1/ r_i, q_i/ f_{max}$. Ce problème a été démontré NP-difficile au sens fort dans (Garey et Johnson, 1976) et (Lenstra et al., 1977). Il a fait l'objet de plusieurs études. Les plus connues sont dues à (Bratley et al, 1973), (Baker et Su, 1974), (McMahon et Florian, 1975), (Lageweg et al., 1976), (Potts, 1980), (Carlier, 1982) et (Grabowski et al., 1986). Il a été l'objet d'un état de l'art réalisé par (Lageweg et al., 1976). L'algorithme de Jackson noté JS (Jackson Schedule) est l'heuristique polynomiale la plus célèbre proposée pour la résolution de ce problème. Il permet d'obtenir une solution qui est à une distance inférieure à $\max_{i=1...n} p_i$ de la solution optimale. En se basant sur cet algorithme, Carlier a proposé dans (Carlier, 1982) une procédure par séparation et évaluation (PSE) efficace qui calcule une solution optimale en un temps relativement court pour un nombre de travaux arrivant à 10000. Cette PSE sera présentée en détails dans la section 3.

La contrainte de préemption est notée « pmtn ». Elle permet d'interrompre l'exécution d'un travail (ou tâche) pour exécuter un autre travail considéré plus prioritaire. En ajoutant cette contrainte, le problème est noté $1/ pmtn, r_i, q_i/ f_{max}$, il devient polynomial. En effet, il suffit d'appliquer l'algorithme de Jackson en intégrant la préemption des travaux pour obtenir une solution optimale en un temps polynomial. Cet algorithme est noté dans ce cas par JPS (Jackson Preemptive Schedule).

En ajoutant la contrainte no-idle le problème est noté $1/ pmtn, r_i, q_i,$ no-idle$/ f_{max}$ lorsque les travaux sont préemptifs et $1/ r_i, q_i,$ no-idle$/ f_{max}$ lorsque les travaux sont non préemptifs. Ces deux problèmes ont été abordés dans la littérature par Chrétienne (Chrétienne, 2005), qui a introduit la contrainte no-idle et a discuté de la complexité de certains problèmes à une machine en ajoutant cette contrainte. Il a montré qu'il est parfois possible d'étendre les algorithmes proposés pour résoudre des problèmes à une machine sans la contrainte no-idle pour les résoudre en ajoutant cette contrainte. Il a démontré que le problème $1/ r_i, q_i,$ no-idle$/ f_{max}$ est fortement NP-difficile. D'après (Chrétienne, 2005), les problèmes à une machine ont été abordés en ajoutant la contrainte no-idle essentiellement dans le cas où chaque travail J_i à une date de fin au plus tard d_i et la fonction objectif minimise une combinaison du retard maximum et de l'avance maximum. La contrainte no-idle peut être relâchée en minimisant le nombre d'intervalles de temps-morts machine ou la somme des temps-morts. Parmi ces travaux, notons ceux de Baptiste dans (Baptiste, 2006) qui s'est intéressé à la réduction du nombre d'intervalles de temps-morts machine.

3. CONCEPTS DE BASE

Dans cette section nous présentons les algorithmes JS et JPS et nous décrivons la PSE de Carlier ainsi que quelques résultats théoriques concernant JS. Nous commençons par introduire la représentation d'une solution.

3.1 Représentation d'une solution

Une solution peut être représentée par un graphe où chaque nœud représente un travail et si un travail i précède immédiatement un travail j alors il y a un arc de précédence allant du nœud i au nœud j avec un poids égal à p_i. Il y a deux nœuds supplémentaires dans le graphe qui sont le nœud début noté 0 et le nœud fin noté *. Il y a un arc entre le nœud 0 et chacun des nœuds représentant un travail i, i = 1…n. L'arc allant de 0 à i est valué par r_i. Il y a aussi un arc liant le nœud * et chacun des travaux i, i = 1…n. Chaque arc allant du noeud i au noeud * est valué par $(p_i + q_i)$. Une solution consiste à attribuer une date t_i à chaque travail i. La valeur attribuée à t_i correspond à la valeur d'un plus long

chemin allant du nœud 0 au noeud i. La valeur attribuée à t_0 est nulle. La valeur attribuée à t_* correspond à la valeur d'un plus long chemin allant du nœud 0 au nœud *. Un tel chemin est dit « chemin critique » car sa longueur correspond à f_{max}. Nous avons ainsi $t_* = f_{max}$. Ce chemin passe à travers un sous-ensemble de nœuds i où chaque nœud i représente un travail i de I. Chaque travail i appartenant à un tel chemin est dit « travail critique ».

Une solution peut aussi être représentée par un diagramme de Gantt où l'axe des abscisses représente le temps et l'axe des ordonnées représente les machines (ou les travaux). La représentation des solutions utilisant un diagramme de Gantt par machine met en évidence l'occupation de la machine par les différents travaux ainsi que les intervalles de temps où la machine est libre ou temps-morts. Par contre, la représentation des solutions utilisant un graphe permet de représenter plus clairement le chemin critique.

3.2 Algorithme de Jackson

La règle de Jackson associe un ordre de priorité à chaque travail de telle sorte que le travail qui a la plus grande valeur de q_i est prioritaire pour être exécuté en premier sur la machine. L'algorithme de Jackson utilise cette règle pour résoudre le problème 1/ r_i, q_i/ f_{max} et calcule, en un temps polynomial, une solution à moins de $\max_{i=1...n} p_i$ de la solution optimale (Carlier, 1982). Soit,

- t : l'instant courant,
- I : l'ensemble de tous les travaux i = 1…n,
- U : l'ensemble des travaux ordonnancés à l'instant t,
- $\overline{U}$: l'ensemble des travaux non encore ordonnancés à l'instant t avec, $\overline{U} = I \setminus U$,
- R : l'ensemble des travaux qui sont prêts pour être exécutés à l'instant t avec, $R = \{i / r_i \leq t, i \in \overline{U}\}$,
- $\overline{R}$: l'ensemble des travaux non encore ordonnancés et qui ne sont pas prêts pour être exécutés à l'instant t avec, $\overline{R} = \overline{U} \setminus R$,
- Q : l'ensemble des travaux prêts pour être exécutés dont la valeur de q_i est maximale avec, $Q = \{j \in R / q_j = \max_{i \in R} q_i\}$, et
- C : la date de fin de traitement de tous les travaux sur la machine.

L'algorithme de Jackson utilise cette règle pour résoudre le problème 1/ r_i, q_i/ f_{max}. Il peut être décrit comme suit :

Algorithme de Jackson (JS)
Début

$t = \min_{i \in I} r_i$; U = Ø ; $\overline{U} = I$; R = Ø ; $\overline{R} = \overline{U}$;

Tant que ($\overline{U} \neq$ Ø) faire
  Q = Ø ;
  Pour tout i ∈ $\overline{R}$ faire
    Si ($r_i \leq t$) alors $R = R \cup \{i\}$; $\overline{R} = \overline{R} \setminus \{i\}$;
    Fin si
  Fin pour
  Pour tout i ∈ R faire
    Si ($q_i = \max_{j \in R} q_j$) alors $Q = Q \cup \{i\}$; Fin si
  Fin pour
  Choisir arbitrairement un travail i ∈ Q
  Ordonnancer i : $t_i = t$;
  $U = U \cup \{i\}$; $\overline{U} = \overline{U} \setminus \{i\}$; $R = R \setminus \{i\}$;
  Mettre à jour t : $t = \max(t_i + p_i , \min_{j \in \overline{U}} r_j)$;
Fin tant que
Fin

3.2.1 Exemple illustratif

Considérons l'exemple dont les données sont présentées dans le tableau 1.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$\mathbf{r_i}$	10	13	11	20	30	0	40
$\mathbf{p_i}$	5	6	7	4	3	6	2
$\mathbf{q_i}$	7	26	24	21	8	17	0

Tableau 1. Données du problème de l'exemple

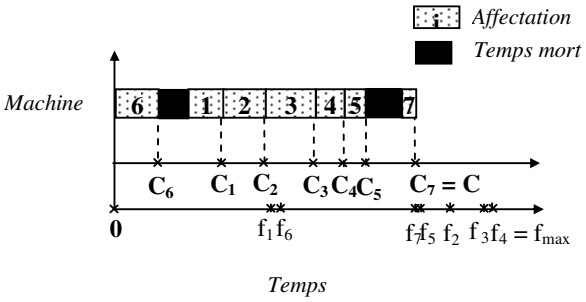

Figure 1. Diagramme de Gantt associé à σ_{JS} de l'exemple décrit dans le tableau 1

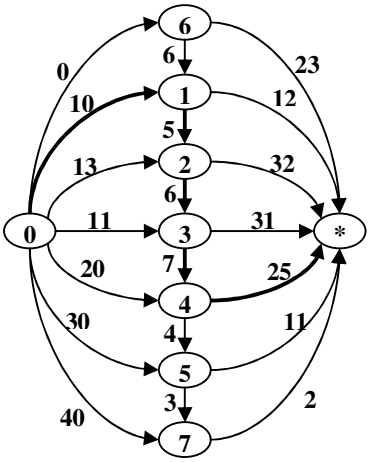

Figure 2. Graphe conjonctif associé à σ_{JS} de l'exemple décrit dans le tableau 1

Soit σ_{JS} la solution construite par l'algorithme JS. L'ordre d'exécution des travaux relatif à σ_{JS} de l'exemple décrit dans le tableau 1 est 6, 1, 2, 3, 4, 5, 7 avec $t_0 = t_6 = 0$, $t_1 = 10$, $t_2 = 15$, $t_3 = 21$, $t_4 = 28$, $t_5 = 32$, $t_7 = 40$ et $t_* = f_{max}(\sigma_{JS}) = 53$. La représentation graphique de cette solution utilisant un diagramme de Gantt est rapportée dans la figure 1 alors que celle utilisant un graphe conjonctif est illustrée dans la figure 2.

3.2.2 Optimalité de JS

Carlier a étudié l'optimalité de JS et a démontré la proposition suivante :

Proposition 1 (Carlier, 1982) :

$\forall$ I' $\subseteq$ I, $h(I') = \min_{i\in I'} r_i + \sum_{i\in I'} p_i + \min_{i\in I'} q_i$ est une borne inférieure de la valeur du f_{max} optimale pour le problème $1/ r_i, q_i/ f_{max}$.

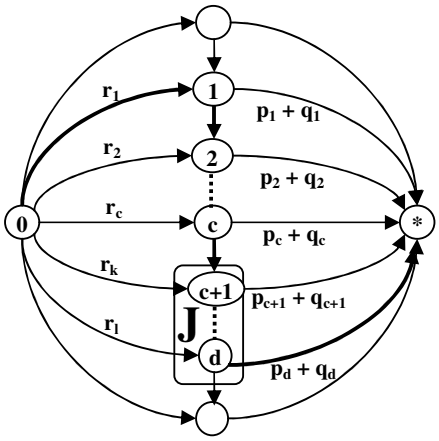

Figure 3. Définitions de c et J

Théorème 1 (Carlier, 1982) :

Soit L la valeur de f_{max} associée à σ_{JS}.

(a) Si σ_{JS} est optimale alors il existe J $\subseteq$ I telle que h(J) = L.

(b) Si σ_{JS} n'est pas optimale alors il existe un travail critique c et un ensemble de travaux critiques J n'incluant pas c telle que $h(J) > L - p_c$. D'où, la distance de σ_{JS} par rapport à l'optimum est inférieure à p_c et, dans une solution optimale, le travail critique c s'exécutera soit avant soit après l'ensemble critique J.

Dans le graphe associé à σ_{JS}, notons μ un chemin critique de longueur maximale. Supposons, sans perte de généralité, que μ passe à travers un sous-ensemble maximum de travaux i. Supposons que μ passe à travers les travaux 1, 2, …, d ($d \leq n$), dans cet ordre. S'il n'existe pas un travail critique c telle que $q_c < q_d$ alors $q_d = \min_{i=1...d} q_i$ et dans ce cas $L = h(\mu)$. D'où L est optimale. Sinon, le choix d'un travail critique c dans μ peut se faire de telle sorte que c soit l'indice du premier travail précédant d telle que $q_c < q_d$ et le choix d'un ensemble de travaux critiques se fait par J = {c + 1, …, d}. Dans ce cas, nous avons $h(J) > L - p_c$. Ainsi, L est à une distance inférieure à p_c par rapport à la solution optimale et le travail c peut être exécuté soit avant soit après l'ensemble des travaux critiques de J.

3.2.3 La PSE de Carlier

La PSE de Carlier est une méthode arborescente performante qui permet d'obtenir une solution optimale au problème $1/r_i, q_i/ f_{max}$. Cette méthode se base sur l'algorithme JS, sur le travail critique c et sur l'ensemble de travaux critiques J. Elle est décrite comme suit : A chaque nœud η de l'arbre est associé un problème à une machine, une borne inférieure $B(\eta)$, et une borne supérieure f_0 qui est la valeur de la meilleure solution connue en arrivant à η. $B(\eta)$ et f_0 sont initialisées dans le premier nœud généré η_0. Nous considérons le nœud de l'arbre qui a la plus petite borne inférieure et nous appliquons l'algorithme JS au problème à une machine associé à ce nœud.

Soit μ le chemin critique associé à σ_{JS} de η. Si $f_{max}(\eta) = h(\mu)$ alors la solution est optimale. Sinon, d'après le théorème 1, il existe un travail critique c et un ensemble de travaux critiques J telle que $h(J) > L - p_c$. D'où, la distance par rapport à la solution optimale est inférieure à p_c et, dans une solution optimale, le travail critique c est exécuté, soit avant, soit après les travaux de J. Le choix d'un travail critique c et d'un ensemble de travaux critiques J se fait comme indiqué précédemment.

Deux nouveaux problèmes sont alors considérés :

1- Dans le premier problème, le travail c est exécuté après tous les travaux de J. Donc, le travail c ne doit pas débuter avant la date $\min_{i\in J} r_i + \sum_{i\in J} p_i$. D'où, $r_c = \max(r_c, \min_{i\in J} r_i + \sum_{i\in J} p_i)$.

2- Dans le deuxième problème, le travail c est exécuté avant tous les travaux de J. Donc le travail c doit avoir une durée de latence q_c supérieure à $\sum_{i\in J} p_i + \min_{i\in J} q_i$. D'où, $q_c = \max(q_c, \sum_{i\in J} p_i + \min_{i\in J} q_i)$.

La nouvelle borne inférieure prend le maximum entre $B(\eta)$, h(J) et h ($J \cup \{c\}$). Un nouveau nœud est ajouté à l'arbre si sa borne inférieure est inférieure à la valeur de la meilleure solution connue f_0.

Soit, $K = \{i/\ i \in I \setminus J \text{ et } p_i > f_0 - h(J)\}$. Si $i \in K$ alors dans un ordonnancement optimal, le travail i est exécuté soit avant soit après les travaux de J. Les deux tests suivant permettent d'améliorer l'efficacité de la PSE :

Test 1 : Si $i \in K$ et $r_i + p_i + \sum_{k\in J} p_k + q_d > f_0$ alors dans un ordonnancement optimal, le travail i est exécuté après les travaux de J et ainsi $r_i = \max(r_i, \min_{k\in J} r_k + \sum_{k\in J} p_k)$.

Test 2 : Si $i \in K$ et $\min_{k\in J} r_k + p_i + \sum_{k\in J} p_k + q_i > f_0$ alors dans un ordonnancement optimal, le travail i est exécuté avant les travaux de J et ainsi $q_i = \max(q_i , \min_{k\in J} q_k + \sum_{k\in J} p_k)$.

Le chemin critique associé à σ_{JS} de l'exemple est formé par l'ensemble J des travaux 1, 2, 3, 4 avec $h(J) = 39 \neq f_{max}(\sigma_{JS}) = 53$. Cette solution n'est pas optimale. En appliquant la PSE de Carlier, 3 nœuds ont été générés avant de déduire la solution optimale σ_{opt} suivante : 6, 3, 2, 4, 1, 5, 7 avec $f_{max}(\sigma_{opt}) = 50$.

3.3 Algorithme de Jackson préemptif

Dans le cas du problème 1/ pmtn, r_i, q_i/ f_{max}, il est possible d'interrompre l'exécution d'un travail sur la machine pour exécuter un autre travail plus prioritaire. L'utilisation de la règle de Jackson a permis de résoudre ce problème et d'obtenir une solution optimale en un temps polynomial. Or, il ne suffit pas de favoriser parmi les travaux qui sont prêts pour être exécutés, ceux dont la valeur de q_i, est maximale, il faut aussi que l'exécution du travail en cours soit interrompue chaque fois qu'un autre travail plus prioritaire est prêt. Le travail interrompu est repris ultérieurement lorsqu'aucun autre travail plus prioritaire n'est prêt pour être exécuté.

Carlier a démontré dans (Carlier, 1982) que la valeur du f_{max} optimale pour le problème 1/ pmtn, r_i, q_i/ f_{max} est égale à $\max_{I'\subseteq I} h(I')$. Notons que la valeur de f_{max} optimale pour le problème 1/ pmtn, r_i, q_i/ f_{max} est une borne inférieure pour celle relative au problème 1/ r_i, q_i/ f_{max}.

Soit σ_{JPS} la solution obtenue en appliquant l'algorithme JPS. L'ordre d'exécution des travaux relatif à σ_{JPS} de l'exemple est 6, 1, 3, 2, 3, 4, 1, 5, 1, 7 avec $t_6 = 0$, $t_1 = 10$, $t_3 = 11$, $t_2 = 13$, $t_3 = 19$, $t_4 = 24$, $t_1 = 28$, $t_5 = 30$, $t_1 = 33$, $t_7 = 40$ et $f_{max}(\sigma_{JPS}) = 49$.

Dans ce qui suit nous proposons d'étendre l'algorithme JPS pour résoudre le problème 1/ pmtn, r_i, q_i, no-idle/ f_{max} puis nous étendons l'algorithme JS afin d'adapter la PSE de Carlier pour résoudre le problème 1/ r_i, q_i, no-idle/ f_{max}.

4. RÉSOLUTION DU PROBLÈME NON-IDLE AVEC PRÉEMPTION

Dans cette section, nous nous intéressons à la résolution du problème 1/ pmtn, r_i, q_i, no-idle/ f_{max}. L'application de l'algorithme JPS au problème 1/ pmtn, r_i, q_i/ f_{max} permet d'obtenir une solution optimale. Or, cette solution peut contenir des temps-morts. Pour avoir une solution no-idle, il suffit de retarder le début d'exécution de chaque allocation de temps machine d'un travail par la somme des durées de temps-morts qui le suivent. Or, cette opération peut augmenter la valeur de f_{max}. En effet, après cette opération la valeur de f_{max} correspondante à σ_{JPS} de l'exemple passe de 49 à 54. Dans ce cas on ne peut pas garantir que la valeur de f_{max} obtenue est optimale.

L'algorithme JPS calcule une date de fin minimale d'utilisation de la machine. Soit C la valeur de cette date. Pour éviter la présence de temps-morts il faut que la machine soit inactive jusqu'à la date $C - \sum_{k=1}^{n} p_k$. Pour cela, il faut augmenter la valeur de la date de début au plus tôt r_i associée à un travail i si celle-ci est inférieure à $C - \sum_{k=1}^{n} p_k$. Nous proposons alors d'attribuer à chaque travail i une nouvelle date de début au plus tôt $\bar{r}_i$ telle que $\bar{r}_i = \max (r_i , C - \sum_{k=1}^{n} p_k)$. Cette nouvelle date nous permet de vérifier la contrainte no-idle. Pour résoudre le problème 1/ pmtn, r_i, q_i, no-idle/ f_{max} nous pouvons appliquer l'algorithme suivant qui est noté NIJPS. La solution obtenue par cet algorithme est notée σ_{NIJPS}.

Algorithme NIJPS
Début
Calculer C en appliquant l'algorithme JPS
Pour i = 1 à n faire
$\bar{r}_i = \max (r_i , C - \sum_{k=1}^{n} p_k)$ Fin pour
Appliquer l'algorithme JPS au problème 1 / pmtn, $\bar{r}_i$, q_i/ f_{max}
Fin

La solution obtenue en appliquant NIJPS à l'exemple est telle que : $t_6 = 9$, $t_2 = 15$, $t_3 = 21$, $t_4 = 28$, $t_5 = 32$, $t_1 = 35$, $t_7 = 40$ et $f_{max}(\sigma_{NIJPS}) = 53$.

Lemme 1 :
Les deux problèmes 1/ pmtn, $\bar{r}_i$, q_i, no-idle/ f_{max} et 1/ pmtn, r_i, q_i, no-idle/ f_{max} partagent le même ensemble de solutions réalisables.

Démonstration
Soit P le problème 1/ pmtn, r_i, q_i, no-idle/f_{max} et P' le problème 1/ pmtn, $\bar{r}_i$, q_i, no-idle/ f_{max}. Soit σ une solution réalisable pour P. Nécessairement la machine terminera son activité à une date C' postérieure à la date C. La solution étant no-idle, la machine débute alors son activité à la date $C' - \sum_{k=1}^{n} p_k$ donc après la date $C - \sum_{k=1}^{n} p_k$ ($C' \geq C$). En conséquence, aucun travail i ne s'exécute avant sa date de début au plus tôt $\bar{r}_i$. Donc, la solution σ est réalisable pour P'. La réciproque est vraie.

Proposition 2 :
L'algorithme NIJPS donne une solution optimale pour le problème 1/ pmtn, r_i, q_i, no-idle/ f_{max}.

Démonstration
La solution σ_{NIJPS} obtenue en appliquant NIJPS au problème 1/ pmtn, $\bar{r}_i$, q_i/ f_{max} est optimale pour ce problème. Dans cette solution, les travaux sont ordonnancés au plus tôt sur la machine et un travail interrompu est repris dès que la machine soit libre. Aussi, $\forall i \in I$ nous avons $\bar{r}_i = \max(r_i , C - \sum_{k=1}^{n} p_k)$ ce qui implique que la machine doit débuter le traitement d u premier travail à la date $t_0 = \min_{i \in J} \bar{r}_i = C - \sum_{k=1}^{n} p_k$.
Soit $t = C_i = t_i + p_i$ la date de fin de traitement d'un travail i. Supposons qu'à la fin de traitement de ce travail i, il n'y a aucun travail qui est prêt pour être exécuté sur la machine alors deux cas se présentent :
Cas 1 : i est l'indice du dernier travail traité sur la machine ;
Cas 2 : la machine doit attendre pendant un intervalle de temps $\delta > 0$ que le prochain travail soit prêt pour être exécuté sur la machine.
Soit $\bar{U}$ l'ensemble des travaux non encore ordonnancés à la date t. Or, $\forall j \in \bar{U}$ nous avons $C_j > C - \sum_{k=1}^{n} p_k$ ce qui implique que $\forall j \in \bar{U}$ nous avons $\bar{r}_j = r_j$. Par conséquent, la machine doit finir le traitement des travaux à une date $C' \geq t_i + p_i + \delta + \sum_{k \in \bar{U}} p_k > C$. D'où la date C' est la date de fin minimale d'utilisation de la machine pour le problème 1/ pmtn, r_i, q_i/ f_{max}. Ceci est en contradiction avec la minimalité de C. Ainsi, l'algorithme NIJPS calcule une date de fin minimale d'utilisation de la machine qui est égale à C. D'où la solution obtenue en appliquant l'algorithme NIJPS est no-idle donc optimale pour le problème 1/ pmtn, $\bar{r}_i$, q_i, no-idle/ f_{max}. Or, les deux problèmes 1/ pmtn, $\bar{r}_i$, q_i, no-idle/ f_{max} et 1/ pmtn, r_i, q_i, no-idle/ f_{max} partagent le même ensemble de solutions réalisables (lemme 1). D'où cette solution est optimale pour le problème1/ pmtn, r_i, q_i, no-idle/ f_{max} .

Etant donnée que la valeur du f_{max} optimale pour le problème 1/ pmtn, r_i, q_i/ f_{max} correspond à $\max_{I' \subseteq I} h(I')$.
Nous pouvons ainsi déduire la proposition suivante :

Proposition 3 :
La durée de la solution obtenue avec NIJPS est égale à $\max_{I' \subseteq I} \bar{h}(I')$ où $\bar{h}(I') = \min_{i \in I'} \bar{r}_i + \sum_{i \in I'} p_i + \min_{i \in I'} q_i$.

Démonstration
Les deux problèmes 1 / pmtn, $\bar{r}_i$, q_i , no-idle / f_{max} et 1/ pmtn, r_i, q_i, no-idle/ f_{max} partagent le même ensemble de solutions réalisables. Donc, ils sont équivalents (Proposition 2).

Le problème 1/ pmtn, r_i, q_i, no-idle/ f_{max} est polynomial et il suffit de modifier les dates de début au plus tôt pour obtenir une solution optimale no-idle. Cette solution est une borne inférieure pour le problème 1/ r_i, q_i, no-idle/ f_{max}.

5. RÉSOLUTION DU PROBLÈME NON-IDLE SANS PRÉEMPTION

Le problème 1/ r_i, q_i, no-idle/ f_{max} est fortement NP-difficile (Chrétienne, 2005). Pour sa résolution, nous proposons d'étendre l'algorithme JS et d'adapter la PSE de Carlier. En effet, la solution obtenue par l'algorithme JS peut contenir des temps-morts comme le montre la figure 1. Ceci est dû au fait que chaque travail a une date de début au plus tôt et que la machine peut être libre alors qu'il n y a pas un travail prêt pour être exécuté. Si les travaux ont la même date de début au plus tôt, alors toutes les solutions réalisables et ordonnancées au plus tôt seraient no-idle. Par conséquent la solution optimale serait no-idle. Donc en ajoutant la contrainte no-idle les deux problèmes seraient équivalents puisqu'ils partagent le même ensemble de solutions réalisables et ils partageront aussi le même ensemble de solutions optimales car ils optimisent les mêmes critères.

5.1 Extension de l'algorithme JS

L'algorithme JS permet de déterminer une solution au problème 1/ r_i, q_i / f_{max} à moins de $\max_{i=1...n} p_i$ de la solution optimale. D'autre part, cet algorithme permet de calculer une date de fin minimale d'utilisation de la machine. Soit C la valeur de cette date. Pour obtenir une solution no-idle en appliquant JS, il faut que la machine débute le traitement des travaux à une date supérieure ou égale à $C - \sum_{i=1}^{n} p_i$. Pour cela, il faut modifier les dates de début au plus tôt de chaque travail i par $\bar{r}_i$ si $r_i < C - \sum_{k=1}^{n} p_k$.
D'où, $\forall$ i = 1…n, $\bar{r}_i = \max (r_i , C - \sum_{k=1}^{n} p_k)$.

Lemme 2 :
Les deux problèmes 1/ $\bar{r}_i$, q_i, no-idle/ f_{max} et 1/ r_i, q_i, no-idle/ f_{max} partagent le même ensemble de solutions réalisables.

Démonstration
La preuve est similaire à celle du lemme 1.

Nous proposons d'étendre l'algorithme JS. Soit l'algorithme NIJS l'extension proposée pour JS.

Algorithme NIJS
Début
Calculer C en appliquant JS
Pour i = 1 à n faire
$\bar{r}_i = \max (r_i , C - \sum_{k=1}^{n} p_k)$ Fin pour
Appliquer JS au $1/ \bar{r}_i , q_i / f_{max}$
Fin

Proposition 4 :
L'algorithme NIJS donne une solution no-idle pour le problème $1/ \bar{r}_i , q_i$, no-idle/ f_{max}. Si cette solution est optimale, alors elle l'est aussi pour le problème $1/ r_i, q_i$, no-idle/ f_{max}.

Démonstration
La solution σ_{NIJS} obtenue en appliquant NIJS au problème $1/ \bar{r}_i , q_i/ f_{max}$ est no-idle (même démonstration que celle pour la proposition 2). Donc, σ_{NIJS} est réalisable pour le problème $1/ \bar{r}_i , q_i$, no-idle/ f_{max} et si elle est optimale pour le problème $1/ \bar{r}_i , q_i/ f_{max}$, alors elle est optimale pour le problème $1/ \bar{r}_i , q_i$, no-idle/ f_{max} . Or, les problèmes $1/ r_i, q_i$, no-idle/ f_{max} et $1/ \bar{r}_i , q_i$, no-idle/ f_{max} partagent le même ensemble de solutions réalisables (lemme2). D'où, si cette solution est optimale pour le problème $1/ \bar{r}_i , q_i$, no-idle/ f_{max}, alors elle est optimale pour le problème $1/ r_i, q_i$, no-idle/ f_{max}.

5.2 Adaptation de la PSE de Carlier

Les résultats du théorème 1 sont vrais pour le problème $1/\bar{r}_i , q_i/ f_{max}$. Soit σ_{NIJS} la solution construite en appliquant NIJS et soit μ_{NIJS} un chemin critique associé à σ_{NIJS}. Soit J_{NI} un ensemble de travaux critiques successifs dans μ_{NIJS}, c_{NI} un travail critique appartenant à μ_{NIJS} mais pas à J_{NI} et L_{NI} la valeur de f_{max} associée à σ_{NIJS}. Si σ_{NIJS} n'est pas optimal alors il existe un travail critique c_{NI} et un ensemble de travaux critiques J_{NI} non incluant c_{NI} telle que $h(J_{NI}) > L_{NI} - p_{c_{NI}}$. Le choix de c_{NI} et de J_{NI} se fait comme dans la PSE de Carlier. D'où, la distance de σ_{NIJS} par rapport à l'optimum est inférieure à $p_{c_{NI}}$ et dans une solution optimale, le travail critique c_{NI} s'exécutera soit avant soit après l'ensemble critique J_{NI}. D'où, l'algorithme NIJS détermine une solution au problème $1/\bar{r}_i , q_i/ f_{max}$ à moins de $\max_{i=1...n} p_i$ de la solution optimale. Pour résoudre le problème $1/ r_i, q_i$, no-idle/ $f_{max,}$ nous proposons d'appliquer la PSE de Carlier en faisant appel à l'algorithme NIJS au lieu de l'algorithme JS. La solution trouvée serait no-idle et elle optimise le problème $1/ \bar{r}_i , q_i$, no-idle/ f_{max}. Par conséquent, cette solution est optimale pour le problème $1/ r_i, q_i$, no-idle/ f_{max} (proposition 4). Or, si σ_{NIJS} est optimale alors elle est no-idle. D'où, elle est optimale pour $1/ \bar{r}_i , q_i$, no-idle/ f_{max} et par la suite, elle est optimale pour $1/r_i, q_i$, no-idle/ f_{max} et nous avons $h(J_{NI}) = L_{NI}$. Mais if faut réajuster $\bar{r}_i$ après chaque modification de $\bar{r}_{c_{NI}}$, pour que la solution obtenue après application de JS soit no-idle. Donc, en remplaçant l'appel de JS dans la PSE de Carlier par NIJS, la solution optimale obtenue est no-idle.

A chaque nœud η de l'arbre est associé un problème à une machine, une borne inférieure $B(\eta)$, et une borne supérieure f_0 qui est la valeur de la meilleure solution connue en arrivant à η. $B(\eta)$ et f_0 sont initialisées dans le premier nœud généré η_0. Nous considérons le nœud de l'arbre qui a la plus petite borne inférieure et nous appliquons l'algorithme NIJS au problème à une machine associé à ce nœud. S'il existe un travail critique c_{NI} et un ensemble de travaux critiques J_{NI} dont le choix se fait comme indiqué précédemment, alors deux nouveaux problèmes sont considérés. La nouvelle borne inférieure prend le maximum entre $B(\eta)$, $h(J_{NI})$ et $h(J_{NI} \cup \{c_{NI}\})$. Un nouveau nœud est ajouté à l'arbre si et seulement si sa borne inférieure est inférieure à la valeur de la meilleure solution connue f_0.
Supposons, sans perte de généralité, que μ_{NIJS} passe à travers les travaux 1, 2, …, d_{NI} ($d_{NI} \leq n$), dans cet ordre. Le choix d'un travail critique c_{NI} dans μ_{NIJS} se fait donc de telle sorte que c soit l'indice du premier travail précédant d_{NI} telle que $q_{c_{NI}} < q_{d_{NI}}$ et le choix d'un ensemble de travaux critiques se fait par $J = \{c_{NI} + 1, …, d_{NI}\}$. Le PSE de Carlier est une procédure de recherche arborescente qui peut être décrit comme suit :

Algorithme PSE de Carlier adaptée
Début
Appliquer NIJPS au problème 1/pmtn, r_i, q_i, no-idle/ f_{max} pour initialiser la borne inférieure $B(\eta)$
$B(\eta) = f_{max}(\sigma_{NIJPS})$; $f_0 = 0$;
Appliquer la procédure récursive générer arbre au problème $1/ r_i, q_i$, no-idle/ f_{max}
Fin

La génération de l'arbre de recherche est comme suit :

Procédure générer arbre
Début
Appliquer l'algorithme NIJS au problème $1/ r_i, q_i$, no-idle/ f_{max} ;
Si (($f_0 = 0$) ou ($f_0 > f_{max}(\sigma_{NIJS})$) alors $f_0 = f_{max}(\sigma_{NIJS})$; $\sigma_0 = \sigma_{NIJS}$; Fin si
Si ($B(\eta) < h(\mu_{NIJS})$) $B(\eta) = h(\mu_{NIJS})$;
Si ($B(\eta) < f_0$) alors

Ajouter un nouveau noeud η à l'arbre
S'il existe un travail critique c_{NI} alors
Si $(B(\eta) < h(J))$ alors $B(\eta) = h(J)$; fin si
Si $(B(\eta) < h(J_{NI} \cup \{c_{NI}\}))$ alors $B(\eta) = h(J_{NI} \cup \{c_{NI}\})$; Fin si
Si $(f_{max}(\sigma_{NIJS}) > h(J))$ alors
Si $(r_{c_{NI}} + p_{c_{NI}} + \sum_{k \in J_{NI}} p_k + q_{d_{NI}} > f_0)$ alors
$r_{c_{NI}} = \max(r_{c_{NI}}, \min_{k \in J_{NI}} r_k + \sum_{k \in J_{NI}} p_k)$;
Appliquer la procédure générer arbre pour le nouveau problème généré en modifiant $r_{c_{NI}}$;
Fin si
Si$(\min_{k \in J_{NI}} r_k + p_{c_{NI}} + \sum_{k \in J_{NI}} p_k + q_{d_{NI}} > f_0))$ alors
$q_{c_{NI}} = \max(q_{c_{NI}}, \min_{k \in J_{NI}} q_k + \sum_{k \in J_{NI}} p_k)$;
Appliquer la procédure solution générer arbre pour le nouveau problème généré en modifiant $q_{c_{NI}}$; Fin si
Si non $f_0 = f_{max}(\sigma_{NIJS})$; $\sigma_0 = \sigma_{NIJS}$; Fin si
Si non $f_0 = f_{max}(\sigma_{NIJS})$; $\sigma_0 = \sigma_{NIJS}$; Fin si Fin si
Fin

Exemple :
Pour obtenir une solution optimale no-idle pour l'exemple précédent, nous appliquons la PSE de Carlier en faisant appel en chaque nœud à NIJS au lieu de JS. Partant de σ_{NIJS} comme suit : 6, 2, 3, 4, 5, 1, 7 avec $t_0 = t_6 = 9$, $t_2 = 15$, $t_3 = 21$, $t_4 = 28$, $t_5 = 32$, $t_1 = 35$, $t_7 = 40$ et $t_* = 53$. Le chemin critique associé à cette solution est formé par l'ensemble J de travaux 6, 2, 3, 4. $h(J) = 49 \neq f_{max} = 53$. Cette solution n'est pas optimale. Il faut alors appliquer la PSE de Carlier pour trouver une solution optimale. En appliquant la PSE de Carlier, il a fallu générer 3 nœuds pour arriver à la solution optimale 3, 2, 4, 6, 5, 1, 7 avec $f_{max} = 51$.

6. TESTS ET RÉSULTATS

La PSE de Carlier et la PSE de Carlier adaptée au problème en ajoutant la contrainte no-idle ont été implémentées en langage C. Des tests ont été générés aléatoirement selon une distribution uniforme pour un nombre de travaux n = 50, 100, 150, …, 1000, 2000, …, 5000. Au total 1584 tests ont été exécutés sur un ordinateur portable TOSHIBA intel(R) pentium(R) M (1,60 GHz, 512 Mo de RAM). Pour chaque test les données relatives à chacun des travaux i = 1…n sont générées aléatoirement comme suit : une date de début au plus tôt r_i est générée aléatoirement entre 1 et r_{max} , une durée d'exécution p_i est générée aléatoirement entre 1 et p_{max}, et enfin une durée de latence q_i est générée aléatoirement entre 1 et q_{max} avec $p_{max} = 50$ et $r_{max} = q_{max} = (1/50)*n*p_{max} *K$. Pour chaque valeur de n, 66 tests sont générés pour K = 1, 2, 3 …, 45 ainsi que pour K = 50, …, 100 et pour K = 110, 120, …, 200.

Dans ce qui suit, nous proposons de comparer les résultats des deux PSE selon le nombre de nœuds explorés. Soit, Nmoy(n), Nmax(n) et Nmin(n) respectivement le nombre moyen, maximal et minimal de nœuds explorés pour résoudre un problème généré avec n tâches sans ajouter la contrainte. Ces valeurs sont notées respectivement par N'moy(n), N'max(n) et N'min(n) en ajoutant la contrainte no-idle. Nous comparons les résultats des deux PSE aussi selon le temps d'exécution mis en millisecondes. Soit, tmoy(n), tmax(n) et tmin(n) le temps d'exécution moyen ainsi que maximal et minimal mis pour résoudre un problème généré avec n tâches sans ajouter la contrainte no-idle. Ces valeurs sont notées respectivement par t'moy(n), t'max(n), et t'min(n) en ajoutant la contrainte no-idle. Ces différentes valeurs ont été calculées pour chaque valeur de K et sont notées respectivement par Nmoy(K), Nmax(K), Nmin(K), N'moy(K), N'max(K), N'min(K), tmoy(K), tmax(K) et tmin(K), t'moy(K), t'max(K) et t'min(K).

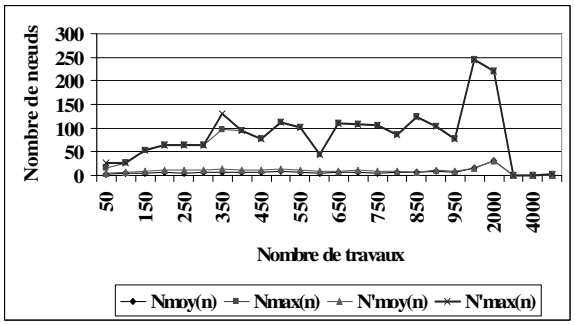

Figure 4. Variations des nombre de nœuds explorés maximal et moyen en fonction de n

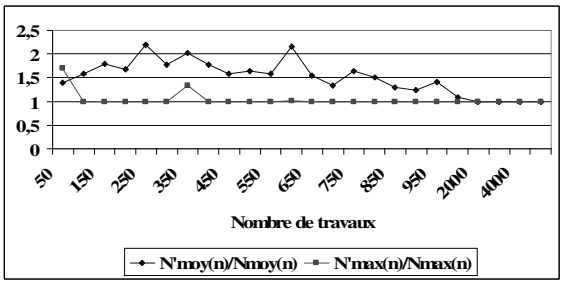

Figure 5. Variations des rapports des nombres de nœuds explorés en fonction de n

Les résultats numériques obtenus concernant les comparaisons en nombre de nœuds sont rapportés dans les figures 4 et 5. Ils peuvent être résumés comme suit :

- Le nombre N'max(n) varie entre 27 et 245 avec une moyenne égale à 85,29 alors que le nombre Nmax(n) varie entre 16 et 245 avec une moyenne égale à 83,42.
- Le nombre N'moy(n) varie entre 3,95 et 31,23 avec une moyenne égale à 9,76 alors que le nombre Nmoy(n) varie entre 2,83 et 31,18 avec une moyenne égal à 6,76.

- Le nombre N'min(n) est égal à 1 et il est atteint pour chaque valeur de n. De même pour N'min(n).
- A partir de n = 3000, le nombre de nœuds explorés est égale à 1 quelle que soit la valeur de K et ceci dans les deux cas avec et sans ajouter la contrainte no-idle.
- Le rapport N'max(n)/Nmax(n) varie entre 1 et 1,69 avec une moyenne égale à 1,04.
- Le rapport N'moy(n)/Nmoy(n) varie entre 1 et 2,2 avec une moyenne égale à 1,51.

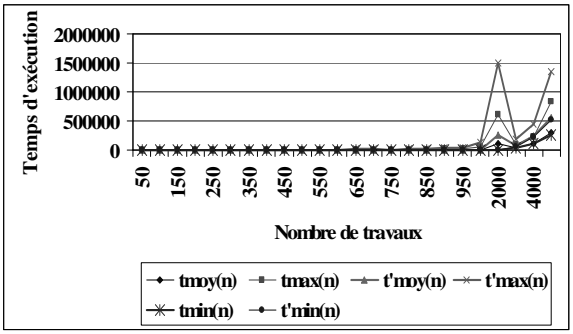

Figure 6. Variations des temps d'exécution (ms) minimal, moyen et maximal en fonction de n

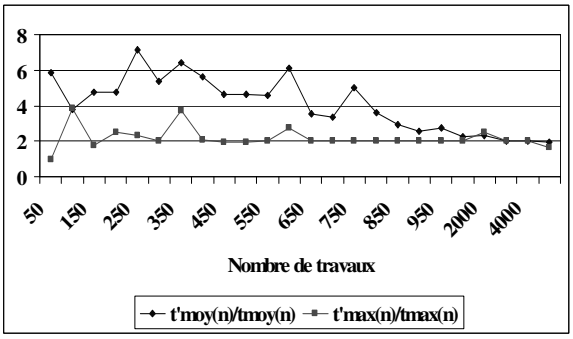

Figure 7. Rapports des variations des temps d'exécution (ms) maximal et moyen en fonction de n

Les résultats numériques obtenus concernant les comparaisons en temps d'exécution sont rapportés dans les figures 6 et 7. Ils peuvent être résumés comme suit :

- Le temps d'exécution t'moy(n) varie entre 1,41 et 562263,26 avec une moyenne égal à 47969,11 alors que le temps d'exécution tmoy(n) varie entre 0,24 et 289625,03 avec une moyenne égale à 23414,93.
- Le temps t'max(n) varie entre 16 et 1505469 avec une moyenne égal à 157822,33 alors que le temps d'exécution tmax(n) varie entre 16 et 833594 avec une moyenne égale à 79176,46.
- Le temps d'exécution t'min(n) varie entre 0 et 511110 avec une moyenne égal à 33007,75 alors que le temps d'exécution tmin(n) varie entre 0 et 254328 avec une moyenne égale à 16255,04.
- Le rapport t'moy(n)/tmoy(n) varie entre 1,94 et 7,16 avec une moyenne égale à 4,09.
- Le rapport t'max(n)/tmax(n) varie entre 1 et 3,88 avec une moyenne égale à 2,18.

D'après ces résultats, nous remarquons que :

- Le nombre maximal de nœuds atteint en ajoutant la contrainte no-idle est pratiquement le même que celui trouvé sans ajouter la contrainte no-idle.
- Le temps d'exécution nécessaire pour résoudre un problème avec n tâches est généralement supérieur à celui nécessaire pour le résoudre sans ajouter la contrainte no-idle.
- Le temps d'exécution maximal nécessaire pour résoudre un problème généré avec n tâches est en moyenne égale à 2,18 fois celui nécessaire pour le résoudre sans ajouter la contrainte no-idle.
- Le temps d'exécution moyen nécessaire pour résoudre un problème généré avec la contrainte no-idle est en moyenne égale à 4,09 fois celui nécessaire pour le résoudre sans ajouter la contrainte no-idle.

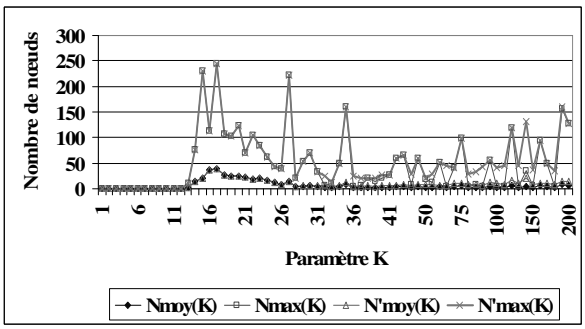

Figure 8. Variation du nombre de nœuds explorés maximal, minimal et moyen en fonction de K

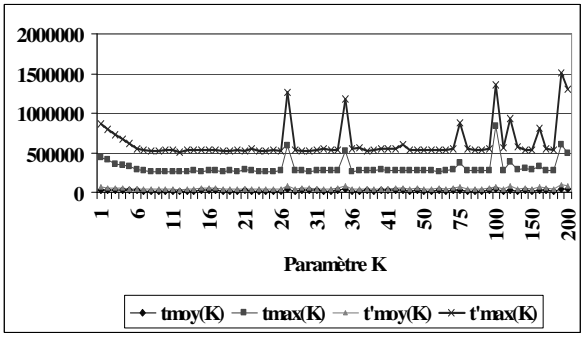

Figure 9. Variations des temps d'exécution en millisecondes en fonction de K

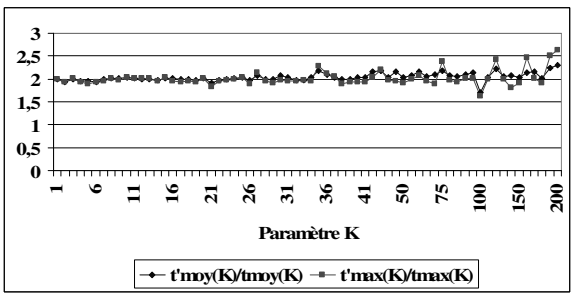

Figure 10. Variations des rapports des temps d'exécution en millisecondes en fonction de K

Nous avons étudié la variation du nombre de nœuds minimal, maximal et moyen en fonction de K. Nous avons remarqué que ce nombre est légèrement plus grand en ajoutant la contrainte no-idle. En effet, d'après la figure 8, le nombre maximal de nœuds explorés sans ajout de la contrainte no-idle varie pour K allant de 14 à 30 entre 20 et 245 avec une moyenne égale à 104,06. En ajoutant la contrainte no-idle, il varie pour les mêmes valeurs de K entre 20 et 245 avec la même moyenne égale à 104,06. Le nombre maximal de nœuds explorés sans ajout de la contrainte no-idle varie pour K allant de 35 à 200 entre 4 et 161 avec une moyenne égale à 42,31 et en ajoutant la contrainte no-idle, il varie pour les même valeurs de K entre 21 et 161 avec une moyenne égale à 57,75. La résolution des problèmes générés est considérée facile (en un seul nœud) pour une valeur de K allant de 1 à 13 et ceci avec et sans ajout de la contrainte no-idle.

D'après la figure 9 et 10, les problèmes générés et résolus en ajoutant la contrainte no-idle ont un temps d'exécution plus élevé que si la contrainte no-idle n'a pas été ajoutée.

En conclusion les comparaisons montrent qu'en ajoutant la contrainte no-idle le problème devient le plus souvent au moins aussi difficile à résoudre. En effet, un problème généré avec n tâches nécessite au moins le même nombre de nœuds pour être résolu. Or, ceci n'est pas toujours vrai car dans certains cas en ajoutant la contrainte no-idle le nombre de nœuds explorés est plus petit et le temps d'exécution mis est plus court.

7. CONCLUSION

Dans cet article, nous avons étudié l'influence de l'ajout de la contrainte no-idle à un problème à une machine où chaque travail i a une date de début au plus tôt r_i et une durée de latence q_i. Nous avons étudié le cas où les travaux sont préemptifs, nous avons montré qu'il est possible d'obtenir une solution optimale en un temps polynomial utilisant la règle de Jackson. Puis nous avons étudié le cas où les travaux sont non préemptifs et nous avons montré qu'il est possible d'obtenir une solution optimale en utilisant la procédure par séparation et évaluation (PSE) proposée par Carlier pour la résolution du problème $1/r_i$, q_i/f_{max}. Nous avons implémenté cette procédure en langage C et nous l'avons adapté au problème en ajoutant la contrainte no-idle afin qu'elle offre une solution optimale no-idle. Les tests ont montré que l'efficacité de la procédure a légèrement diminué en ajoutant la contrainte no-idle. En effet, le nombre de nœuds explorés et le temps d'exécution augmentent. Nous pouvons conclure que l'algorithme reste comme même efficace puisqu'il permet d'offrir une solution optimale en un temps relativement court. La résolution de ce problème en ajoutant la contrainte no-idle peut être intéressante pour la résolution de problèmes à une machine similaires mais en ajoutant d'autres contraintes ou d'autres critères comme elle peut servir pour la résolution de problèmes plus complexes où l'atelier comprend plusieurs machines.

RÉFÉRENCES

Baker K. R. and Z. S. Su, 1974. Sequencing with due-dates and early starts times to minimize maximum tardiness. *Naval Research Logistics Quarterly,* 21(1), p. 171-176.

Bratley P., M. Florian and P. Robillard, 1973. On sequencing with earliest starts and due dates with application to computing bounds for the n/m/g/f_{max} problem. *Naval Research Logistics Quarterly,* 20(1), p. 57-67.

Baptiste P., 2006. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. *Proceedings on the seventeenth annual ACM-SIAM symposium on Discrete algorithms, January (22-26), Miami, Florida,* p. 364-367.

Brucker P., 1995. Scheduling Algorithms. Springer. ISBN : 3-540-60087-6.

Carlier J., 1982. The one-machine sequencing problem. *European Journal of Operational Research,* 11, p. 42-47.

Chrétienne P., 2005. On the No-Wait Single-Machine Scheduling Problem. *In the 7th Conference on Models and Algorithms for planning and scheduling, Juin,* p. 76-79.

Garey M. R. and D. S. Johnson, 1976. Computers and intractability: a guide to the theory of NP-completeness. *W.H. Freeman and Co, San Francisco, California*.

Grabowski J., E. Nowicki and S. Zdrzalka, 1986. A bloc approach for single machine scheduling with release dates and due dates. *European Journal of Operational Research,* 26, p. 278-285.

Jackson J.R., 1955. Scheduling a production to minimize maximum tardiness. *Research Report No. 43, Management Science Research Report, University of California, Los Angeles*.

Lageweg B. J., J.K. Lenstra, and A.H.G. Rinnooy Kan, 1976. Minimizing maximum lateness on one machine: computational experience and some applications. *Statistica Neerlandica,* 30, p. 25-41.

Lenstra J.K., A.H.G. Rinnooy Kan and P. Brucker, 1977. Complexity machine scheduling problems. *Annals of Discrete Mathematics*, 1, p. 343-362.

McMahon G.B. and M. Florian, 1975. On scheduling with ready times and dues dates to minimize maximum lateness. *Operations Research,* 23(3), p. 475-482.

Potts C.N., 1980. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research,* 28 (6), p. 1436-1441.