Une procédure pour la résolution du problème de jobshop cyclique

Touria Ben Rahhou [*,**] **Martin Fink** [*,***] **Laurent Houssin** [*,**]

[*] *CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France*
[**] *Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France*
{*mfink , ben-rahhou , houssin @laas.fr* }
[***] *TUM-School of Management, Technische Universität München, 80333 Munich, Germany*

Résumé : Nous nous intéressons, dans cet article, aux problèmes d'atelier à cheminements multiples, communément appelés "jobshop", et plus précisément à la version cyclique de ce problème, avec comme objectif, la minimisation du temps de cycle asymptotique de l'ordonnancement. Ainsi, une procédure de séparation et d'évaluation permettant la résolution exacte de ce genre de problème est proposée. Cette procédure est basée sur la théorie des graphes, elle utilise la consistance des graphes pour déduire des bornes pour les décalages événementiels et l'algorithme de Howard pour l'évaluation des solutions faisables. Nous comparons ensuite cette nouvelle procédure à d'autres méthodes exactes et présentons les résultats numériques de ces comparaisons.

Mots-clés: Ordonnancement cyclique, Optimisation discrète, Algorithme de "Branch and bound", Problème de Jobshop.

1. INTRODUCTION

Dans un problème d'ordonnancement classique, un ensemble de tâches est exécuté une seule fois, l'ordonnancement établi optimise les fonctions objectifs telles que la minimisation du makespan ou la minimisation des pénalités d'avance et de retard. En revanche, dans un problème d'ordonnancement cyclique, un ensemble de tâches dites génériques est exécuté une infinité de fois, l'objectif étant de minimiser le temps entre deux occurrences d'une même tâche. L'ordonnancement cyclique intervient dans de nombreux domaines, comme la robotique (voir Kats et Levner (1997)), l'industrie (voir Pinedo (2005) et Hillion et Proth (1989)) ou de la programmation multiprocesseurs (voir Hanen et Munier (1995b)). Plusieurs points de vue ont été utilisés pour aborder cette classe de problèmes. La plupart d'entre eux tire parti de la théorie des graphes, de la programmation linéaire en nombres entiers, des réseaux de Petri et de l'algèbre $(max, +)$. Un aperçu des problèmes d'ordonnancement cyclique et des différentes approches est disponible dans Hanen et Munier (1995a).

Nous considérons le problème de jobshop cyclique (CJSP), qui est un problème NP-difficile (voir Hanen (1994)) et utilisons la théorie des graphes pour le représenter (voir Brucker et Kampmeyer (2008) et Hanen et Munier (1995a)). L'objectif de la résolution d'un CJSP est de trouver un ordonnancement périodique ayant un temps de cycle asymptotique minimal et vérifiant toutes les contraintes de précédence et de ressource. Deux méthodes de résolution exactes existent pour résoudre ce problème. La première, un programme linéaire en nombre entier décrit dans Brucker et Kampmeyer (2008) et Hanen (1994). La deuxième, une procédure de séparation et d'évaluation décrite dans Hanen (1994).

Dans ce papier, une nouvelle procédure de séparation et d'évaluation est proposée, elle fixe les bornes des décalages événementiels entre les différentes occurrences de tâches en utilisant la consistance d'un graphe, et évalue ensuite les solutions possibles en utilisant la version algèbre $(max, +)$ de l'algorithme de Howard (voir Cochet-Terrasson et al. (1998)). Par la suite, cette approche est comparée à deux autres procédures pour évaluer ses performances.

La deuxième partie de ce papier présente le problème d'ordonnancement cyclique de base et cite brièvement les algorithmes pour le résoudre. Par la suite, le CJSP est défini et les deux méthodes de résolution aux quelles nous allons nous comparer sont décrites. Dans la troisième partie, nous introduisons la nouvelle procédure de "branch and bound" pour la résolution de CJSP. Ensuite, dans la quatrième partie, les résultats de comparaisons des différentes méthodes sont donnés et discutés. Enfin, dans la dernière partie, nous concluons et présentons quelques perspectives de notre étude.

2. PROBLÈME : DÉFINITIONS ET NOTATIONS

Nous présentons d'abord le problème d'ordonnancement cyclique de base (GBCSP) et quelques algorithmes pour le résoudre. Ensuite, un rappel des différentes définitions et notations sur le problème de jobshop cyclique (CJSP) est fait et les caractéristiques des deux méthodes de résolution exactes qui vont être comparées à notre procédure sont présentées enfin de cette partie.

2.1 Problème d'ordonnancement cyclique de base

Le GBCSP est caractérisé par un ensemble de tâches élémentaires $\mathcal{T} = \{1, ..., n\}$ qui seront répétées une infinité de fois. Chaque tâche i a une durée p_i, on note $< i, k >$ la $k^{ème}$ occurrence de i. Dans un GBCSP, les tâches sont liées par des contraintes de précédence dites uniformes. Résoudre un GBCSP revient à déterminer pour chaque occurrence $< i, k >$, sa date de début $t(i, k)$. Si on considère un ordonnancement cyclique avec un temps de cycle α, on obtient :

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N} : \quad t(i, k) = t(i, 0) + \alpha k. \tag{1}$$

Sachant que l'occurrence $< i, k + 1 >$ ne peut pas commencer avant la fin de l'occurrence $< i, k >$, on peut écrire la constraint suivante :

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N} : \quad t(i, k + 1) \geq t(i, k) + p_i. \tag{2}$$

Un graphe orienté $G = (T, E)$ avec T un ensemble de sommets et E un ensemble d'arcs peut être associé à un GBCSP tel que : un sommet (resp. un arc) de G correspond à une tâche élémentaire (resp. une contrainte uniforme) dans le GBCSP. Chaque arc (i, j) de G est doublement valué : la première valeur est la longueur $L_{ij} \in \mathbb{R}$ et la deuxième valeur est la hauteur $H_{ij} \in \mathbb{Z}$. On exprime les contraintes uniformes comme suit :

$$\forall (i, j) \in E, \forall k \in \mathbb{N} : \quad t(i, k) + L_{ij} \leq t(j, k + H_{ij}). \tag{3}$$

2.2 Méthodes de résolution d'un GBCSP

Le graphe G permet de vérifier la consistance du GBCSP en utilisant le théorème suivant (Voir Hanen (1994), Carlier et Chrétienne (1988), Cohen et al. (1985)) :

Théorème 1. Un GBCSP est consistant si et seulement si chaque circuit du graphe associé G a une hauteur positive.

Par ailleurs, le temps de cycle minimal d'un GBCSP est donné par le circuit critique du graphe associé. Plusieurs algorithmes existent pour trouver le circuit critique d'un graphe orienté doublement valué.

Nous pouvons d'abord citer l'algorithme de Gondran et Minoux (1995) qui a une complexité en $O(n^3 log(n))$. Un deuxième algorithme est celui de Karp de complexité $O(n^3)$ (voir Karp (1978); Dasdan et Gupta (1998)), il ne fonctionne que lorsque toutes les hauteurs H_{ij} qui existent sont égales à 1. Quand $H_{ij} \geq 0$, quelques modifications sont nécessaires (voir Baccelli et al. (1992)). Quant au cas où $H_{ij} \leq 0$ fréquemment rencontré dans les GBCSP (*cf.* l'exemple de base de Hanen (1994)), une première tentative pour les prendre en compte dans l'algorithme de Karp a été faite dans Houssin (2011). Le troisième algorithme est celui de Howard (voir Howard (1960)), initialement conçu pour les processus décisionnels de Markov, il a été adapté à l'algèbre $(max, +)$ dans Cochet-Terrasson et al. (1998). Bien que la complexité de l'algorithme de Howard reste non prouvée, il montre d'excellentes performances et un temps de calcul presque linéaire.

2.3 Problème de Jobshop cyclique

La différence majeure entre le CJSP et le GBCSP est que, dans un GBCSP, chaque tâche est excécutée sur sa propre machine dédiée. Alors que dans le CJSP, le nombre de machines est inférieur au nombre de tâches. Par conséquent, les machines jouent un rôle important et les contraintes qu'elles génèrent sont ajoutées au CJSP. Un CJSP est donc un GBCSP auquel on rajoute des contraintes de ressources. Nous définissons un ensemble de machines $\mathcal{M} = \{1, ..., m\}$ où $m < n$. La tâche i est exécutée sur la machine $M(i) \in \mathcal{M}$ sans interruption pendant toute sa durée p_i. Nous considérons le chevauchement de deux occurrences de tâches différentes utilisant la même machine, comme étant un conflit de ressources. Ce conflit est évité en rajoutant des contraintes disjonctives. On note T_s l'ensemble de tâches utilisant la machine $s \in \mathcal{M}$. Les contraintes disjonctives peuvent s'écrire comme suite :

$$\forall s \in \mathcal{M},\ \forall i, j \in T_s,\ \forall k, l \in \mathbb{N} :$$
$$t(i, k) \leq t(j, l) \Rightarrow (t(i, k) + p_i \leq t(j, l)) \ or \ (i = j \ and \ k = l). \tag{4}$$

Comme pour les GBCSP, un graphe orienté $G = (T, E)$ peut être associé au CJSP tel que : un sommet (resp. un arc) de G correspond à une tâche élémentaire (resp. contrainte) dans le CJSP. Tout arc uniforme (i, j) de G possède deux valeurs $L_{ij} = p_i$ et H_{ij}. Tout arc disjonctif a aussi deux valeurs : la durée $L_{ij} = p_i$ et le décalage événementiel $K_{ij} \in \mathbb{Z}$. Pour toute machine s, si $i \in T_s$ et $j \in T_s$ alors : $K_{ij} + K_{ji} = 1$ (voir Hanen (1994) pour plus de détails). Deux sommets factices sont rajoutés au graphe, ils représentent le début et la fin d'une occurrence k. L'arc qui relie ces deux sommets est également doublement valué avec une longueur nulle et une hauteur non négative. Cette hauteur représente le nombre maximal de travaux en cours, plus d'explications sur cette valeur de la hauteur sont disponibles dans Brucker et Kampmeyer (2008).

Exemple Un ensemble de quatre tâches élémentaires regroupées en deux travaux doit être exécuté de façon cyclique sur deux machines. Le tableau 1 indique le détail de ce CJSP et la figure 1 représente le graphe associé. Les sommets 1 (début) and 6 (fin) sont des sommets factices.

Table 1. Données de l'exemple de CJSP

Travail	1		2	
Tâche	2	3	4	5
Durée	5	4	2	3
Machine	1	2	1	2

2.4 Méthodes de résolution d'un CJSP

Un CJSP tel qu'il est décrit dans le paragraphe 2.3 peut être résolu avec les deux méthodes exactes suivantes :
- Le programme linéaire en nombres entiers (MILP) décrit dans Hanen (1994) ou dans Brucker et Kampmeyer (2008).
- La procédure de séparation et d'évaluation proposée par Hanen (1994).

Dans la suite, nous résumons brièvement les principales caractéristiques de ces deux approches.

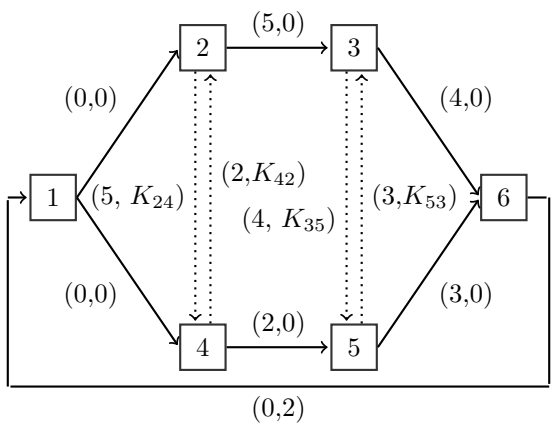

FIGURE 1. Graphe associé de l'exemple de CJSP

Programme linéaire en nombres entiers Un CJSP peut être transformé en un MILP. Quelques modifications sont nécessaires car les contraintes de ressources engendrent une contrainte non-linéaire avec $\alpha \times K_{ij}$. Ainsi, le débit $\tau = \alpha^{-1}$ est introduit avec une nouvelle variable $u_i = t_i \times \alpha^{-1}$, ce qui conduit au programme linéaire suivant.

$$\max \tau \tag{2.5}$$
$$\text{s.t.}$$
$$u_j - u_i \geq \tau \times p_i - H_{ij} \tag{2.6}$$
$$u_j - u_i \geq \tau \times p_i - K_{ij} \tag{2.7}$$
$$K_{ij} + K_{ji} = 1 \tag{2.8}$$

Procédure de Branch and Bound proposée par Hanen Dans Hanen (1994), l'auteur présente une procédure de séparation et d'évaluation basée sur deux concepts, les fonctions conservatrices de hauteur et les amplitudes. L'auteur prouve qu'une fonction conservatrice de hauteur faisable est associée à un ensemble d'ordonnancements faisables, parmi lesquels on trouve la meilleure solution, en utilisant l'algorithme de Gondran et Minoux (1995) en $O(n^3 log(n))$. Les amplitudes sont utilisées pour calculer les bornes supérieures des hauteurs.

Une procédure d'ajustement est utilisée pour évaluer les noeuds. Elle augmente, itérativement, les bornes inférieures des amplitudes et baisse les bornes supérieures des hauteurs jusqu'à ce qu'aucun changement ne puisse être fait ou qu'un circuit d'amplitude positive soit détecté et la solution est donc irréalisable. Puisque la procédure de Hanen nécessite une borne supérieure du temps de cycle pour débuter, une heuristique est développé à cet effet dans Hanen (1994).

3. NOUVELLE PROCÉDURE DE SÉPARATION ET D'ÉVALUATION POUR LE CJSP

Nous proposons une procédure de branch and bound pour résoudre les CJSP, elle repose sur deux piliers principaux : la consistance d'un graphe pour obtenir des bornes sur les décalages événementiels et l'algorithme de Howard pour calculer les temps de cycle et évaluer un noeud. Le paragraphe suivant établit d'abord les concepts théoriques sous-jacents avant de décrire la structure de l'algorithme. Ensuite, la procédure est appliqué à l'exemple présenté dans le paragraphe 2.3.

3.1 Concepts théoriques

Un graphe G est dit consistant si et seulement si tout circuit C de G a une hauteur plus grande ou égale à 1. La hauteur d'un circuit est, par définition, la somme des hauteurs (ou décalages événementiels) de tous les arcs de ce circuit. Un nouvel arc disjonctif ajouté au graphe crée au moins un circuit et ne compromet pas la consistance du graphe si la solution reste faisable. Par conséquent, chaque arc disjonctif (i, j) a une borne inférieure évidente de son décalage événementiel K_{ij} qui garantit la consistance du graphe.

Appliquons ce concept à notre exemple, les tâches 2 et 4 sont exécutées sur la même machine, deux arcs disjonctifs (un de 2 à 4 et l'autre de 4 à 2) sont donc nécessaires pour déterminer une priorité entre les deux tâches. L'arc disjonctif qui part de 2 à 4 induit un nouveau cycle $C = (2, 4, 5, 6, 1, 2)$. La hauteur de ce cycle $H(C) = K_{24} + 0 + 0 + 2 + 0$ doit être supérieure à 1 pour que la solution reste réalisable. Ce qui donne la borne inférieure de $K_{24} \geq -1$.

Théorème 2. On considère un CJSP où les tâches i et j sont exécutées sur la même machine. On peut déduire une borne inférieure notée K_{ij}^- pour le décalage événementiel K_{ij} telle que :

$$K_{ij}^- = 1 - \min\{H(\mu) | \mu \text{ chemin de } j \text{ à } i \text{ dans } G\}. \tag{3.1}$$

Preuve 1. Une solution réalisable nécessite une hauteur au moins égale à 1 pour tous circuit du graph c.à.d : $\forall C \in G : H(C) \geq 1$. Donc, pour tout arc (i, j), on obtient $K_{ij} + \{H(\mu) | \mu \text{ chemin de } j \text{ à } i \text{ dans } G\} \geq 1$, donc $K_{ij} \geq 1 - \{H(\mu) | \mu \text{ chemin de } j \text{ à } i \text{ dans } G\}$. Cette inégalité est vraie pour tous les chemins de j à i dans G, on prend donc le plus court chemin.

Corollaire 1. Puisque $K_{ij} + K_{ji} = 1$, un intervalle pour chaque variable K_{ij} est déduit comme suit :

$$K_{ij}^- \leq K_{ij} \leq 1 - K_{ji}^- \tag{3.2}$$

Pour avoir un moyen efficace de calcul de bornes pour tous les K_{ij}, nous construisons une matrice des hauteurs B de taille $n \times n$. Ses coefficients B_{ij} sont initialisés par H_{ij} pour les arcs uniformes et par K_{ij} pour les arcs disjonctifs déjà déterminés. Par la suite, l'algorithme de Floyd et Warshall (voir Cormen et al. (2009)) de compléxité $O(n^3)$ est utilisé pour trouver toutes les paires de chemins les plus courts dans B. Les bornes inférieures sont données par l'égalité $K_{ij}^- = 1 - B_{ji}$.

La mise à jour la matrice des hauteurs fait appel à l'algorithme de Floyd et Warshall, elle est donc susceptible de ralentir la procédure. Nous proposons ainsi deux versions de l'algorithme : Version 1 qui met à jour les bornes des K_{ij} en chaque noeud de l'arbre de recherche et par conséquent, n'examine pas la faisabilité de la solution. Version 2 qui ne calcule les bornes qu'une seule fois, lorsque l'algorithme est initialisé, mais vérifie la faisabilité de la solution lors de l'évaluation du noeud avec l'algorithme de Howard.

Par ailleurs, le théorème 3 permet de calculer la borne inférieure du temps de cycle d'un CJSP.

Théorème 3. On considère un CJSP et on note α^- la borne inférieure du temps de cycle.

$$\alpha^- = \max\{\lambda_u, \lambda_s\}. \tag{3.3}$$

où λ_u est la solution du GBCSP associé et λ_s qui est définie comme suite :

$$\lambda_s = \max_{m \in \mathcal{M}} \{\sum p_i | M(i) = m\}.$$

Preuve 2. Un CJSP sans contraintes de ressource est équivalent à un GBCSP et il est impossible de trouver un temps de cycle inférieur à λ_u. La deuxième borne inférieure est la somme de toutes les durées de tâches utilisant la même machine. Puisque :

$$t_j(k+1) \geq \sum_{M(i)=M(j)} p_i + t_j(k).$$

On en déduit que le temps de cycle ne peut pas être inférieur à la somme maximale des durées de tâches sur une même machine.

3.2 Structure de la procédure de branch and bound

L'algorithme de séparation et d'évaluation version 1 est présenté dans cette partie. La version 2, elle, est expliquée à travers ses différences avec la première version. L'algorithme peut être divisé en deux parties, à savoir la tête où on initialise la procédure et le corps qui contient la boucle principale.

La tête de l'algorithme consiste en la méthode *initialize()*, qui détermine la borne supérieure du temps de cycle en additionnant toutes les durées de tâches. On crée le premier noeud, on calcule son temps de cycle et on vérifie la faisabilité de la solution initiale. La borne inférieure du temps de cycle est calculée selon le théorème 3. Si le noeud initial est faisable, on l'empile et on détermine les bornes inférieures des décalages événementiels en deux temps : d'abord, on initialise tous les B_{ij} par H_{ij} si l'arc (i, j) existe et par ∞ sinon. Ensuite, on utilise l'algorithme de Floyd et Warshall pour trouver le plus court chemin dans B.

Le corps de l'algorithme répéte un ensemble d'opérations jusqu'à ce que la pile soit vide ou jusqu'à ce que la borne inférieure soit égale à la borne supérieure pour le temps de cycle. La méthode *checkSum()*, comme dans la procédure de Hanen (voir Hanen (1994)), si $K_{ij}^- + K_{ji}^- = 1$. Si tous les K_{ij} sont déterminés, un ordonnancement complet est obtenu. Dans ce cas, la borne supérieure du temps de cycle est mise à jour par le α en cours si $\alpha \leq \alpha^+$. Si $\alpha \geq \alpha^+$, on coupe l'arbre de recherche. Dans le cas d'ordonnancement non complet, un nouveau K_{ij} est sélectionné selon la méthode *branchingRule()*. Cette méthode choisit un K_{ij} non déterminé et pour lequel la somme $K_{ij}^- + K_{ji}^-$ est maximale, ce qui induit le nombre minimal de noeuds fils, et donc réduit l'arbre de recherche. L'algorithme fait ensuite appel à la méthode *branch()* qui crée un nouveau noeud fils pour tout entier dans l'intervalle $[K_{ij}^-, 1 - K_{ji}^-]$ pour le K_{ij} sélectionné. La méthode *evaluateNodes()* met à jour les bornes inférieurs des décalages événementiels en insérant la valeur K_{ij} dans la matrice des hauteurs B et en faisant appel à l'algorithme de Floyd and Warshall, la deuxième étape de la méthode *evaluateNodes()* est d'utiliser l'algorithme de Howard pour mettre à jour le temps de cycle. Enfin, l'algorithme appelle la méthode *nodeSelection()* qui empile tous les noeuds fils dans la pile selon la règle de sélection qui stipule que le noeud qui possède le plus petit temps de cycle est choisi en premier. En cas d'égalité de temps de cycle, on choisit selon les indices de noeuds. La procédure générale est décrite dans la figure 2.

Input: $G = (T, E)$, affectation des tâches aux machines
Output: α
`//Tête`
$S_0 \leftarrow$ *Initialize()*
`//Corps`
while *PileNoeuds* $\neq$ *VIDE and* $\alpha^- \neq \alpha^+$ **do**
  $S \leftarrow$ Le premier élément de PileNeouds
  checkSum(S)
  if $S(\alpha) < \alpha^+$ **then**
    if *S est une séquence complète* **then**
      $\alpha^+ \leftarrow S(\alpha)$
    else
      $S(K_{ij}$ sélectionné$) \leftarrow$ *branchingRule(S)*
      $N \leftarrow$ *branch(S)*
      evaluateNodes(N)
      PileNoeud $\leftarrow$*nodeSelection(N)*
return α^+

FIGURE 2. Algorithme de la nouvelle procédure

Dans la version 2 de l'algorithme, C'est la méthode *evaluateNodes()* qui change : Tout d'abord, la matrice B n'est pas mis à jour, elle est seulement initialisée comme dans la version 1. Cela conduit au deuxième ajustement : la faisabilité de chaque noeud est vérifiée avec l'algorithme de Howard.

3.3 Exemple

Dans ce paragraphe, nous résolvons l'exemple du paragraphe 2.3 avec notre procédure version 1.

La méthode *initialize()* calcule $\alpha^+ = 14$. Elle crée donc le premier noeud S_0 qui représente le graphe uniforme et calcule $\alpha^- = \max\{5; 7\} = 7$. Avant d'empiler S_0, les bornes inférieures de K_{24}, K_{42}, K_{35} et K_{53} sont calculées avec la matrice des hauteurs B représentée dans le tableau 2.

TABLE 2. La matrice des hauteurs B initiale

1	0	0	0	0	0
2	1	0	2	2	0
2	2	1	2	2	0
2	2	2	1	0	0
2	2	2	2	1	0
2	2	2	2	2	1

On sait que $K_{ij}^- = 1 - B_{ji}$. Alors $K_{24}^- = -1$, $K_{42}^- = -1$, $K_{35}^- = -1$, et $K_{53}^- = 1$.

Dans le corps de l'algorithme, on sélectionne S_0 et on l'enlève de la pile de noeuds. La méthode *checkSum()* ne modifie pas la solution car $K_{24}^- + K_{42}^- \neq 1$ et $K_{35}^- + K_{53}^- \neq 1$. Pour S_0, $\alpha = 7$ donc $\alpha \leq \alpha^+$. La méthode *branchingRule()* sélectionne K_{35} plutôt que K_{24} car l'intervalle pour K_{24} est $[-1; 2]$ comparé à $[-1; 0]$. La méthode *branch()* crée deux noeuds S_1 et S_2 avec respectivement les valeurs $(K_{35} = -1,\ K_{53} = 2)$ et $(K_{35} = 0,\ K_{53} = 1)$. Pour les deux noeuds, la méthode *evaluateNodes()* calcule le temps de cycle, respectivement 7 et 9. Ensuite, elle met à jour K_{24}^- et K_{42}^-. Enfin, la méthode *nodeSelection()* choisit S_1. La deuxième itération continue avec S_1. Encore une fois, *checkSum()* ne modifie pas la solution. On choisit K_{24} pour brancher. Ensuite, la méthode *branch()* crée quatre noeuds fils S_3, S_4, S_5 et S_6 avec les valeurs $-1, 0, 1$ et 2 pour K_{24} et *evaluateNodes()* calcule leurs temps de cycle $11, 7, 7$ et 10. *nodeSelection()* choisit $S4$. A la troisième itération, on trouve une solution complète avec $\alpha \leq \alpha^+$ donc on remet à jour $\alpha^+ = \alpha = 7$. Ensuite, la procédure s'arrête car $\alpha^- = \alpha^+$ et le temps de cycle optimal 7 est trouvé.

La figure 3 affiche l'arbre de recherche pour l'exemple.

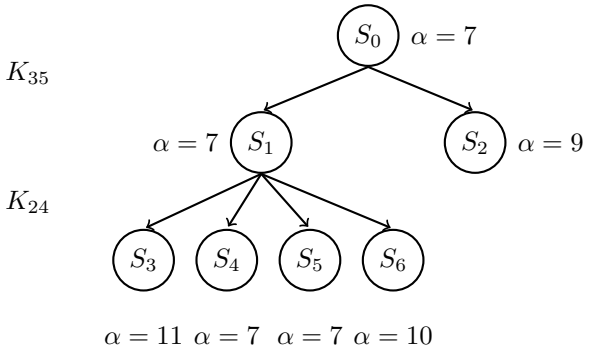

FIGURE 3. Exemple d'arbre de recherche

4. COMPARAISON DE MÉTHODE DE RÉSOLUTION D'UN CJSP

Cette partie est dédiée à l'évaluation de performances. Dans ce but, les deux versions de notre procédure sont testées en utilisant des CJSP générés aléatoirement et les résultats sont comparés avec les performances de la procédure de Hanen et le MILP de Hanen et Brucker.

4.1 Comparaison des différentes procédures de branch and bound

Nous avons comparé trois implémentations de procédures de branch and bound (BnB) pour résoudre le CJSP : notre procédure version 1, notre procédure version 2, la procédure de Hanen avec l'algorithme de Howard. Les différentes caractéristiques sont résumées dans le tableau 3.

En plus des caractéristiques citées dans le tableau 3, notre algorithme fait un branchement en profondeur d'abord et calcule le temps de cycle pour tous les noeuds. L'algorithme de Hanen fait aussi un branchement en profondeur d'abord mais calcule le circuit critique uniquement quand il trouve une solution complète.

TABLE 3. Les différentes caractéristiques des procédures de BnB pour résoudre le CJSP

Procédure	Notre procédure		Hanen
Version	Version 1	Version 2	
Algorithme pour le GBCSP	Howard	Howard	Howard
Calcul des bornes	Consistance du graphe, m.à.j à chaque noeud	Consistance du graphe, vérifié avec Howard	Amplitude
Rejet des noeuds	$\alpha \geq \alpha^+$	$\alpha \geq \alpha^+$, noeud infaisable	Amplitude positive de circuit
Fin de la procédure	Pile de noeuds vide, $\alpha^- = \alpha^+$	Pile de noeuds vide, $\alpha^- = \alpha^+$	Pile de noeuds vide
Sélection de noeud	Plus petit α	Plus petit α	Basé sur les amplitudes de circuit
Règle de branchement	Plus petit intervalle	Plus petit intervalle	Basé sur les amplitudes de circuit

4.2 Caractéristiques techniques

Les tests des quatre implantations ont été effectuées sur un ordinateur avec un AMD Athlon 64 x2 3800+ processor, 1.7 GB RAM et avec Linux Fedora 9 comme système d'exploitation. Les instances de CJSP ont été générées aléatoirement. Elles sont caractérisées par un nombre de travaux, un nombre de tâches et un nombre de machines et suivent l'exemple suivant : Chaque travail est relié à un sommet " début " et un sommet " fin " factices. Ces deux sommets factices sont reliés entre eux par un arc uniforme de longueur (durée) nulle et de hauteur égale à 2. Nous testons six différents types d'instances de CJSP, chacun est composé de dix problèmes, que l'on peut voir dans le tableau 4.

TABLE 4. Aperçu sur les types de problèmes

Type	1	2	3	4	5	6
Nombre de travaux	2	2	5	5	10	8
Nombre de tâches	10	20	10	20	20	30
Nombre de machines	2	5	2	5	2	4

4.3 Résultats numériques

Dans ce paragraphe, nous présentons les résultats numériques des tests pour les quatre implémentations. Pour chacun des six types de problèmes, dix instances sont présentées. La durée de fonctionnement de chaque implémentation est limitée à six secondes. Le tableau 5 reporte le nombre d'instances résolues dans cet intervalle de temps par procédure, et par type de problème.

TABLE 5. Résultats numériques des tests

	1	2	3	4	5	6
Notre procédure version 1	10	10	10	10	9	9
Notre procédure version 2	10	10	10	10	3	5
MILP	10	10	9	8	0	0
Procédure de Hanen	10	10	9	5	0	0

4.4 Interprétation des résultats numériques

Les résultats de tests sont interprétés en deux temps : D'abord, toutes les implémentations sont comparées les unes aux autres, et dans ce cas, trois genres de problèmes se distinguent : problèmes de petites tailles (type 1 et 2), problèmes de tailles moyennes (type 3 et 4) et problèmes de grandes tailles (type 5 et 6). Ensuite, Nous interprétons les différentes performances des deux versions de notre procédure.

Pour problèmes de petites tailles, Les quatre procédures marchent bien et résolvent les 10 problèmes en moins de six secondes. Donc, pour ce genre de problèmes, aucune différence significative n'est observée entre les différentes procédures. La situation change pour les problèmes de tailles moyennes, car notre procédure continue à résoudre toutes les instances, les résultats du MILP avec CPLEX restent satisfaisants, cependant la procédure de Hanen ne résoud que la moitié des instances de problèmes de type 4. La tendance se confirme pour les problèmes de grandes tailles, car notre procédure continue à afficher de bonnes performances, alors que le MILP et la procédure de Hanen n'arrivent plus à résoudre aucune instance en moins de six secondes.

Les deux versions de notre procédure se comportent aussi bien pour les problèmes de petites tailles que pour problèmes de tailles moyennes. Dans le cas des problèmes de grandes tailles, les deux versions diffèrent considérablement. Évidemment, la mise à jour de la matrice des hauteurs prends plus de temps que de laisser l'algorithme de Howard examiner la faisabilité d'un noeud. Mais, moins de noeuds doivent être examinés dans la version 1, comme les bornes inférieures des décalages événementiels ne peuvent qu'augmenter au cours de la procédure et donc l'intervalle et le nombre de noeuds fils diminue.

5. CONCLUSION

Dans ce papier, Nous avons proposé une nouvelle procédure pour résoudre le problème de jobshop cyclique. Ses deux principales caractéristiques sont l'évaluation des noeuds par l'algorithme de Howard et les bornes basées sur la consistance du graphe générique. L'implémentation de deux versions de cette nouvelle procédure est due à utilisation de différentes approches pour la délimitation de l'arbre de recherche. Pour évaluer les performances de nos deux versions, nous les avons comparé à deux autres implémentations de procédures pour la résolution de CJSP : la procédure de branch and bound de Hanen avec l'algorithme de Howard et un MILP résolu avec CPLEX. Les tests ont démontré que pour les problèmes de petites tailles, les trois méthodes se valent. Par contre, pour les problèmes de plus grandes tailles, les performances de notre procédure diffèrent considérablement de ceux des deux autres procédures.

Bien que les résultats soient prometteurs, la nouvelle procédure a encore un potentiel d'amélioration et devrait être soumise à de futures recherches pour obtenir des résultats encore meilleurs, par exemple, l'amélioration la fonction d'évaluation des noeuds ou de la borne inférieure du temps de cycle. En plus de l'amélioration de la procédure elle-même, d'autres expériences de calcul pourraient être effectué, comme des tests sur des instances qui génèrent un très grand nombre d'arcs disjonctifs. Pour être en mesure de faire face à ce genre de problèmes, une présélection d'arcs disjonctifs en utilisant une heuristique pourrait être une solution. En outre, une analyse comparative de notre nouvelle procédure avec la version originale de la procédure Hanen intégrant l'algorithme de Gondran et Minoux devraient être effectuées pour obtenir des comparaisons de performance supplémentaires.

RÉFÉRENCES

Baccelli, F., Cohen, G., Olsder, G.J., et Quadrat, J.P. (1992). *Synchronization and Linearity*. Wiley.

Brucker, P. et Kampmeyer, T. (2008). A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics*, 156(13), 2561 – 2572.

Carlier, J. et Chrétienne, P. (1988). *Problèmes d'Ordonnancement : modélisation, complexité, algorithmes*. Masson, Paris.

Cochet-Terrasson, J., Cohen, G., Gaubert, S., Gettrick, M.M., et Quadrat, J.P. (1998). Numerical computation of spectral elements in max-plus algebra. In *Proceedings of the IFAC Conference on System Structure and Control*. Nantes.

Cohen, G., Dubois, D., Quadrat, J.P., et Viot, M. (1985). A linear system theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Trans. on Automatic Control*, AC–30, 210–220.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., et Stein, C. (2009). *Introduction to Algorithms*. The MIT Press.

Dasdan, A. et Gupta, R.K. (1998). Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(10).

Gondran, M. et Minoux, M. (1995). *Graphes et algorithmes, 3e édition revue et augmentée*. Eyrolles.

Hanen, C. (1994). Study of a np-hard cyclic scheduling problem : The recurrent job-shop. *European Journal of Operational Research*, 72(1), 82 – 101.

Hanen, C. et Munier, A. (1995a). *Scheduling Theory and Its Applications*, chapter Cyclic Scheduling on Parallel Processors : An Overview. Wiley.

Hanen, C. et Munier, A. (1995b). A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57, 167–192.

Hillion, H.P. et Proth, J.M. (1989). Performance evaluation of job-shop systems using timed event graphs. *IEEE Transaction on Automatic Control*, 34(1), 3–9.

Houssin, L. (2011). Cyclic jobshop problem and (max,plus) algebra. In *Proceedings of IFAC WC*. Milan, Italy.

Howard, R. (1960). *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology.

Karp, R.M. (1978). A characterization of the minimum cycle mean in a digraph. *Discrete Math*, 23.

Kats, V. et Levner, E. (1997). A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Operations Research Letters*, 21, 171–179.

Pinedo, M. (2005). *Planning and Scheduling in Manufacturing and Services*. Springer.