RAIRO. RECHERCHE OPÉRATIONNELLE

M.-L. LEVY
P. LOPEZ
B. PRADIN

Décomposition temporelle et caractérisation de solutions admissibles pour le problème d'ordonnancement à une machine

RAIRO. Recherche opérationnelle, tome 33, n° 2 (1999), p. 185-208

<http://www.numdam.org/item?id=RO_1999__33_2_185_0>

RAIRO Rech. Opér.
(vol. 33, n° 2, 1999, pp. 185-208)

DÉCOMPOSITION TEMPORELLE ET CARACTÉRISATION DE SOLUTIONS ADMISSIBLES POUR LE PROBLÈME D'ORDONNANCEMENT À UNE MACHINE

par M.-L. LEVY ([1]), P. LOPEZ ([1]) et B. PRADIN ([2])

Communiqué par Jacques CARLIER

Résumé. – *Nous proposons une approche par caractérisation de solutions admissibles pour la décomposition temporelle du problème d'ordonnancement à une machine avec contraintes de dates limites. La technique de caractérisation repose sur l'utilisation de règles d'*analyse sous contraintes. *Elles permettent de déterminer les positions admissibles de chaque tâche dans une séquence, représentées par des* intervalles de rangs. *Une procédure de décomposition basée sur des comparaisons d'intervalles de rangs est développée pour regrouper les tâches de localisation proche dans toute solution admissible. Des résultats expérimentaux sont présentés pour valider notre approche en tant que phase de préparation à la recherche d'une solution, que celle-ci soit menée par une procédure de résolution automatique, ou par un opérateur humain dans un contexte de coopération homme-ordinateur.*

Mots clés : Ordonnancement, décomposition temporelle, admissibilité.

Abstract. – *We propose a temporal decomposition approach for the one-machine scheduling problem with ready times and due dates. A set of feasible schedules is characterized by means of* constraint-based analysis *rules. They permit the determination of the feasible locations of each task in a sequence, represented by* rank intervals. *A decomposition procedure based on comparisons of rank intervals is developed in order to group tasks to be sequenced in the neighbourhood of each other in a feasible solution. Computational results are reported so that the capability of our approach to prepare the problem for a solution computation, achieved either by a human operator or by an automatic solving procedure, can be evaluated.*

Keywords: Scheduling, temporal decomposition, feasibility.

1. INTRODUCTION

Le problème posé consiste à ordonnancer un ensemble T de n tâches sur une machine ; chaque tâche $i \in T$ est caractérisée par sa date de début au plus tôt r_i, sa date de fin au plus tard d_i et sa durée opératoire p_i. L'intervalle

(*) Reçu en mai 1996.
([1]) LAAS-CNRS, 7, avenue du Colonel Roche, 31077 Toulouse Cedex 4, France.
([2]) INSA, Complexe Scientifique de Rangueil, 31077 Toulouse Cedex 4, France.
e-mail: lopez,pradin@laas.fr

Recherche opérationnelle/Operations Research, 0399-0559/99/02/$ 7.00

de temps $[r_i, d_i]$ représente la *fenêtre temporelle* de la tâche i. On suppose que la relation $d_i - r_i \geq p_i$ est toujours satisfaite $\forall i = 1...n$. Les tâches ne peuvent être interrompues et ne peuvent être réalisées simultanément.

Une solution est donnée par l'ensemble t_i, $i = 1...n$ des dates de début des tâches ; elle est dite *admissible* si elle satisfait les contraintes de fenêtres temporelles et d'utilisation disjonctive de la machine [7]. La recherche d'une solution est généralement guidée par la prise en compte d'objectifs exprimés à l'aide de critères à optimiser. En présence de dates de fin au plus tard, les critères les plus couramment considérés sont liés aux retards, *e.g.*, la minimisation du plus grand retard algébrique $L_{max} = \max_{i \in T}(t_i + p_i - d_i)$ [3].

De nombreux travaux de recherche ont été consacrés au problème à une machine (*voir* par exemple [10] et [18]). Notre attention s'est portée sur ceux qui procèdent par *décomposition temporelle* du problème initial en plusieurs sous-problèmes, plus faciles à appréhender de par leur taille réduite [1, 6, 19]. Dans [6, 19], des procédures de résolution locale et de coordination des sous-problèmes sont mises en œuvre pour calculer une solution approchée vis-à-vis d'un critère à optimiser. À la différence de ces approches, celle de Amamou *et al.* [1] ne cherche pas à déterminer une solution. Elle s'inscrit dans une optique d'aide à la décision et a pour objectif de mettre en évidence des groupes de tâches indépendants dans toute solution admissible; elle a récemment été complétée par une procédure d'énumération partielle de l'ensemble des solutions [11].

Dans ce même esprit de préparation de la phase de recherche d'une solution, nous proposons une approche par décomposition temporelle basée sur la caractérisation de l'ensemble des solutions admissibles. Elle associe les principes de l'*analyse sous contraintes* [7, 8] et le concept d'*intervalle de rangs admissibles* [1] pour fournir une structure décomposée du problème, en accord avec les conditions d'admissibilité des solutions.

L'analyse sous contraintes consiste à rechercher les conditions d'admissibilité déductibles de l'interaction des contraintes. L'association des aspects temporels et séquentiels du problème permet d'envisager la caractérisation des solutions admissibles comme un processus itératif procédant alternativement (1) à des déductions de précédences à partir de l'analyse des relations entre fenêtres temporelles et (2) à des actualisations de dates limites induites par les relations de précédences mises en évidence [7, 9]. La caractérisation complète des solutions admissibles par l'analyse sous contraintes est théoriquement possible mais pose un problème de

complexité que nous abordons en ne recherchant qu'une partie des conditions d'admissibilité, *i.e.*, des *conditions nécessaires*.

L'intervalle de rangs admissibles $\mathcal{R}(i)$ d'une tâche $i \in T$ représente l'ensemble des positions non démontrées interdites pour i dans une séquence admissible. Il est borné par un rang à gauche $R_g(i)$ et un rang à droite $R_d(i)$: $\mathcal{R}(i) = [R_g(i), R_d(i)]$. Les tâches du problème à une machine ne sont soumises à aucune contrainte de précédence ; chaque tâche peut donc initialement occuper les n positions possibles, *i.e.*, $R_g(i) = 1$, $R_d(i) = n$, $\forall i \in T$. Au cours de l'analyse du problème, des actualisations de rangs parallèles aux actualisations de dates limites vont permettre de modéliser les conditions d'admissibilité séquentielles. Comme les conditions d'admissibilité recherchées sont seulement nécessaires, il se peut que les intervalles de rangs contiennent des positions qui ne correspondent à aucune solution admissible. Par ailleurs, la modélisation de l'ensemble des rangs admissibles d'une tâche par un intervalle ne permet pas la mise en évidence de positions interdites situées entre des positions admissibles.

Les principes de base de notre approche pour le problème à une machine [14] peuvent également être appliqués au problème du flow-shop [15]. De nouveaux résultats vont ici être fournis. Nous présenterons en premier lieu les règles d'analyse sous contraintes utilisées et les algorithmes développés pour leur implémentation, en dissociant (1) les précédences issues de l'analyse des fenêtres temporelles par paires et (2) les précédences établies entre une tâche et un ensemble de tâches. Nous définirons ensuite des principes de regroupement basés sur des comparaisons d'intervalles de rangs, puis nous décrirons la procédure de décomposition. Les résultats d'expérimentations seront enfin commentés.

2. PRÉCÉDENCES ENTRE PAIRES

L'analyse des fenêtres temporelles par paires permet de déduire des relations séquentielles du type $i \prec j$; nous les appelons *précédences simples*. Les propositions 1 et 2 se réfèrent à [1].

2.1. Déduction de précédences simples

PROPOSITION 1 : *Soient* $i, j \in T$, $i \neq j$ *et* $m_{i,j} = d_j - r_i - p_i - p_j$. *Si* $m_{i,j} < 0$, *alors il n'existe pas d'ordonnancement admissible où* i *précède* j.

Preuve : Considérons un ordonnancement où i précède j : la date de fin C_j de la tâche j est telle que $C_j \geq r_i + p_i + p_j$. Or, $m_{i,j} < 0$ implique $r_i + p_i + p_j > d_j$, ce qui mène à la contradiction $C_j > d_j$. □

On dit que $m_{i,j}$ est la *marge temporelle* disponible après le séquencement de i avant j. Selon les signes de $m_{i,j}$ et $m_{j,i}$, on peut distinguer quatre cas :

(1) si $m_{i,j} \geq 0$ et $m_{j,i} \geq 0$, alors i peut être avant ou après j ;

(2) si $m_{i,j} \geq 0$ et $m_{j,i} < 0$, alors i doit précéder j ;

(3) si $m_{i,j} < 0$ et $m_{j,i} \geq 0$, alors j doit précéder i ;

(4) si $m_{i,j} < 0$ et $m_{j,i} < 0$, alors le problème ne possède pas de solution admissible.

PROPOSITION 2 : *Soit $i \in T$. Les ensembles $Prec(i)$ des tâches qui précèdent nécessairement la tâche i dans toute solution admissible et $Suiv(i)$ des tâches qui suivent nécessairement la tâche i dans toute solution admissible s'écrivent :*

$$\begin{cases} Prec(i) = \{j \in T \setminus \{i\} \ / \ m_{i,j} < 0 \text{ et } m_{j,i} \geq 0\}, \\ Suiv(i) = \{j \in T \setminus \{i\} \ / \ m_{i,j} \geq 0 \text{ et } m_{j,i} < 0\}. \end{cases}$$

Preuve : Ces expressions sont justifiées par la proposition 1. Les conditions $m_{j,i} \geq 0$ (pour $Prec(i)$) et $m_{i,j} \geq 0$ (pour $Suiv(i)$) permettent d'éliminer les cas d'incohérence. □

ACTUALISATIONS : Lorsque les ensembles $Prec(i)$ et $Suiv(i)$, $i \in T$, sont déterminés, les dates limites et les rangs de la tâche i peuvent être actualisés :

$$Prec(i) \prec i \Rightarrow \begin{cases} r_i \leftarrow \max[r_i, eft(Prec(i))], \\ R_g(i) \leftarrow \max[R_g(i), llr(Prec(i)) + 1], \end{cases} \quad (1)$$

$$i \prec Suiv(i) \Rightarrow \begin{cases} d_i \leftarrow \min[d_i, lst(Suiv(i))], \\ R_d(i) \leftarrow \min[R_d(i), srr(Suiv(i)) - 1], \end{cases} \quad (2)$$

où $eft(Prec(i))$ (resp. $lst(Suiv(i))$) représente la date de fin au plus tôt de $Prec(i)$ (resp. la date de début au plus tard de $Suiv(i)$) et $llr(Prec(i))$ (resp. $srr(Suiv(i))$) désigne la plus grande position nécessairement occupée par une tâche de $Prec(i)$ (resp. la plus petite position nécessairement occupée par une tâche de $Suiv(i)$).

La date de fin au plus tôt de $Prec(i)$ peut être calculée comme la date de fin de l'ordonnancement calé à gauche associé au séquencement des tâches de $Prec(i)$ dans l'ordre croissant de leur date de début au plus tôt [9]. La date de début au plus tard de $Suiv(i)$ est déterminée de façon symétrique. Des procédures analogues permettent de calculer la plus grande et la plus petite position nécessairement occupée par un ensemble de tâches [15].

Dans le cas particulier où $Prec(i) = \{j\}$, $eft(Prec(i)) = r_j + p_j$ et $llr(Prec(i)) = R_g(j)$.

2.2. Algorithme de recherche de précédences simples

La procédure PRÉCPAIRES (Fig. 1) construit, pour chaque tâche $i \in T$, les ensembles $Prec(i)$ et $Suiv(i)$ et effectue les actualisations de dates limites et de rangs qui en résultent pour i. La construction des ensembles $Prec$ et $Suiv$ est de complexité $O(n)$. La complexité des actualisations est celle du tri des ensembles $Prec$ et $Suiv$, soit $O(n \log_2 n)$. Les dates limites des tâches ont des valeurs entières, si bien que le nombre maximal d'actualisations successives que la boucle « Répéter » peut effectuer est λn, avec $\lambda = \max_{i \in T}(d_i - r_i - p_i)$. PrécPaires est donc de complexité théorique $O(\lambda n^3 \log_2 n)$. Il faut toutefois noter que le nombre d'itérations dues à la boucle « Répéter » est en pratique généralement *très* faible.

```
Proc PrécPaires(T)
   Répéter
      PourTout i ∈ T faire
         déterminer les ensembles Prec(i) et Suiv(i);
         actualiser les dates limites et les rangs de i selon les expressions (1)
         et (2);
         Si d_i − r_i < p_i alors arrêt; FinSi {problème incohérent}
      FinPourTout
   Jusqu'à ce qu'il n'y ait plus d'actualisation
FinProc
```

Figure 1. – Algorithme de PRÉCPAIRES.

2.3. Exemple

Considérons l'exemple 1 présenté dans le tableau 1a. Après l'application de la procédure PRÉCPAIRES, les dates limites et les rangs sont ceux du tableau 1b où les valeurs actualisées apparaissent en caractères gras. Trois itérations ont été effectuées par la boucle « Répéter » : deux au cours desquelles des dates limites ou des rangs ont été actualisés et une qui a conclu que plus aucune actualisation n'était déductible.

TABLEAU 1
Ex. 1 - Données initiales (a) et données actualisées par PRÉCPAIRES (b).

i	1	2	3	4	5	6	7
r_i	0	1	1	2	4	5	7
d_i	4	4	5	8	11	12	13
p_i	2	1	1	3	1	3	2

(*a*)

i	1	2	3	4	5	**6**	7
r_{ai}	0	1	1	**4**	4	**7**	7
d_{ai}	4	4	**4**	8	11	12	13
$R_{ga}(i)$	1	1	1	**4**	**4**	**5**	**5**
$R_{da}(i)$	**3**	**3**	**3**	**5**	7	7	7

(*b*)

3. PRÉCÉDENCES ENTRE UNE TÂCHE ET UN ENSEMBLE DE TÂCHES

L'expression des relations séquentielles déductibles entre une tâche i et un ensemble de tâches S nous amène à distinguer (1) les *précédences conjonctives* $i \prec S$ (i précède toutes les tâches de S) et (2) les *précédences non conjonctives* $S \not\prec i$ (i précède au moins une tâche de S). Les propositions 3 et 4 se réfèrent respectivement à [4, 16] et à [9].

3.1. Déduction de précédences conjonctives

PROPOSITION 3 : *Soient* $S \subset T$, $i \in T \setminus S$. *Si*

$$\max_{s \in S} d_s - \min_{s \in S \cup \{i\}} r_s < \sum_{s \in S \cup \{i\}} p_s,$$

alors il n'existe pas d'ordonnancement admissible où i précède une tâche de S, i.e., $S \prec i$.

Preuve : Considérons un ordonnancement de $S \cup \{i\}$ dans lequel i précède une tâche de S. Appelons s_l la dernière tâche à être ordonnancée, C_{s_l} sa date de fin : $C_{s_l} \geq \min_{s \in S \cup \{i\}} r_s + \sum_{s \in S \cup \{i\}} p_s > \max_{s \in S} d_s$. Comme i précède une tâche de S, $s_l \neq i$ et $C_{s_l} > \max_{s \in S} d_s$ mène à une contradiction. □

ACTUALISATIONS : Selon les expressions (1),

$$S \prec i \Rightarrow \begin{cases} r_i \leftarrow \max[r_i, eft(S)], \\ R_g(i) \leftarrow \max[R_g(i), llr(S) + 1]. \end{cases} \quad (3)$$

Les dates de fin au plus tard et les rangs à droite des tâches s de S peuvent également être actualisés (*cf.* expressions (2)) :

$$\forall s \in S, \begin{cases} d_s \leftarrow \min(d_s, d_i - p_i), \\ R_d(s) \leftarrow \min[R_d(s), R_d(i) - 1]. \end{cases} \quad (4)$$

Une proposition symétrique à la proposition 3 permet de déduire des relations du type $i' \prec S'$ et d'actualiser $d_{i'}$, $R_d(i')$, et $\forall s \in S'$, r_s et $R_g(s)$ (*voir* [14]).

Les règles présentées pour la déduction de précédences conjonctives entre une tâche i et un ensemble de tâches S permettent de mettre en évidence certaines précédences non déductibles par l'analyse des relations entre i et chaque tâche de S. Elles ne constituent cependant pas une généralisation des règles présentées en 2.1 car elles ne permettent pas, dans le cas particulier où $S = \{j\}$, de retrouver toutes les précédences simples.

3.2. Déduction de précédences non conjonctives

PROPOSITION 4 : *Soient* $S \subset T$, $i \in T \setminus S$. *Si*

$$\max_{s \in S} d_s - r_i < \sum_{s \in S \cup \{i\}} p_s,$$

alors il n'existe pas d'ordonnancement admissible où i *précède toutes les tâches de* S, *i.e.*, $i \not\prec S$.

Preuve : Considérons un ordonnancement de $S \cup \{i\}$ dans lequel i précède toutes les tâches de S. Appelons s_l la dernière tâche à être ordonnancée, $s_l \in S$ et C_{s_l} sa date de fin ; $C_{s_l} \geq r_i + \sum_{s \in S \cup \{i\}} p_s > \max_{s \in S} d_s$ ce qui conduit à une contradiction. □

Donc, au moins une tâche de S doit précéder i ; selon la terminologie de [9], S est un *faisceau* (non conjonctif) *convergent* pour i.

ACTUALISATIONS :

$$i \not\prec S \Rightarrow \begin{cases} r_i \leftarrow \max[r_i, \min_{s \in S}(r_s + p_s)], \\ R_g(i) \leftarrow \max[R_g(i), \min_{s \in S} R_g(s) + 1]. \end{cases} \quad (5)$$

Une proposition symétrique à la proposition 4 permet de mettre en évidence des *faisceaux* (non conjonctifs) *divergents* $S' \not\prec i'$ et d'actualiser $d_{i'}$ et $R_d(i')$ (*voir* [14]).

Dans le cas particulier où $S = \{j\}$, les propositions 1 et 4 conduisent au même résultat ; les précédences simples sont un cas particulier de faisceaux non conjonctifs.

3.3. Algorithme de recherche de précédences conjonctives et non conjonctives

L'algorithme PRÉCENS que nous proposons pour la recherche de relations de précédences entre une tâche et un ensemble de tâches est présenté figure 2. Il recherche pour chaque tâche $i \in T$ un faisceau convergent *actualisant* $i \not\succ S$ (procédure FAISCCONV), puis teste si une relation conjonctive $\tilde{S} \prec i$, $\tilde{S} \subseteq S$, est vérifiée, et actualise r_i et $R_g(i)$ selon les expressions (3) ou (5) (procédure VÉRIFCONJ&ACTU_R). Seules ces deux procédures sont décrites dans la suite ; un travail symétrique est effectué par les procédures FAISCDIV et VÉRIFCONJ&ACTU_D.

Proc PRÉCENS(T)
  Répéter
    PourTout $i \in T$ faire
      $candidates \leftarrow \{k \in T \setminus \{i\} \ / \ r_k + p_k > r_i$ **ou** $R_g(k) \geq R_g(i)\}$;
      FAISCCONV(i,$candidates$,S);
      Si $S \neq \emptyset$ alors VÉRIFCONJ&ACTU_R(i,S); FinSi
      $candidates' \leftarrow \{k' \in I \setminus \{i\} \ / \ d_{k'} - p_{k'} < d_i$ **ou** $R_d(k') \leq R_d(i)\}$;
      FAISCDIV(i,$candidates'$,S');
      Si $S' \neq \emptyset$ alors VÉRIFCONJ&ACTU_D(i,S'); FinSi
    FinPourTout
  Jusqu'à ce qu'il n'y ait plus d'actualisation
FinProc

Figure 2. – Algorithme de PRÉCENS.

Cette recherche des précédences conjonctives et non conjonctives a l'avantage de ne nécessiter qu'une fois la mise en relation de chaque tâche avec un ensemble de tâches. Il est vrai que la limitation aux faisceaux actualisants ne permet pas de trouver toutes les relations conjonctives actualisantes, mais elle assure que toutes les relations trouvées, conjonctives ou non conjonctives, amèneront une actualisation.

3.3.1. *Procédure* FAISCCONV

La procédure FAISCCONV (Fig. 3) est basée sur l'identification de sous-ensembles de tâches dans lesquels il est inutile de rechercher un faisceau convergent.

PROPOSITION 5 : *Soient* $G \subset T$, $|G| > 1$ *et* $i \in T \setminus G$ *tels que* G *ne soit pas un faisceau convergent pour* i. *Soient* Y *la liste des tâches de* G *classées selon l'ordre décroissant de leur date de fin au plus tard et* S *un sous-ensemble de* G. *On note* l *la position dans* Y *de la tâche de* S *de plus grande date de fin au plus tard, i.e.,* $max_{s\in S} d_s = d_{Y[l]}$. *Si* $\forall\, k = 1...(l-1)$,

$$d_{Y[k]} - p_{Y[k]} \leq d_{Y[k+1]},$$

alors S *ne peut pas être un faisceau convergent pour* i

Preuve : Comme G n'est pas un faisceau convergent pour i, $\max_{g\in G} d_g - r_i \geq \sum_{g\in G} p_g + p_i$. Supposons que S soit un faisceau convergent pour i, on a alors $\max_{s\in S} d_s - r_i < \sum_{s\in S} p_s + p_i$. D'où $\max_{s\in S} d_s - \sum_{s\in S} p_s < \max_{g\in G} d_g - \sum_{g\in G} p_g$. Par ailleurs, $max_{s\in S} d_s = d_{Y[l]}$ signifie que S est un sous-ensemble de $\bigcup_{j=l...|S|} Y[j]$, si bien que $\sum_{s\in S} p_s \leq \sum_{j=l}^{|S|} p_{Y[j]}$. D'où

$$\max_{s\in S} d_s - \sum_{s\in S} p_s \geq \max_{s\in S} d_s - \sum_{j=l}^{|S|} p_{Y[j]}.$$

On déduit :

$$\max_{s\in S} d_s - \sum_{j=l}^{|S|} p_{Y[j]} < \max_{g\in G} d_g - \sum_{g\in G} p_g,$$

i.e., $d_{Y[l]} < d_{Y[1]} - \sum_{j=1}^{l-1} p_{Y[j]}$. Or $\forall\, k = 1...(l-1)$, $d_{Y[k]} - p_{Y[k]} \leq d_{Y[k+1]}$. D'où

$$d_{Y[l]} < d_{Y[1]} - \sum_{j=1}^{l-1} p_{Y[j]} \leq d_{Y[2]} - \sum_{j=2}^{l-1} p_{Y[j]} \leq \cdots$$

$$\leq d_{Y[l-2]} - \sum_{j=l-2}^{l-1} p_{Y[j]} \leq d_{Y[l-1]} - p_{Y[l-1]} \leq d_{Y[l]},$$

ce qui conduit à la contradiction $d_{Y[l]} < d_{Y[l]}$. □

L'algorithme de FAISCCONV évite l'examen de tous les sous-ensembles de *candidates* en exploitant la proposition 5 ; il parcourt la liste des tâches classées en ordre décroissant de leur date de fin au plus tard et ne teste la condition d'existence d'un faisceau (test ①) qu'après avoir supprimé les tâches $Y[k]$ telles que $d_{Y[k]} - p_{Y[k]} \leq d_{Y[k+1]}$ (boucle ②). L'algorithme comporte deux boucles sur l'ensemble *candidates*, avec la suppression d'une tâche à chaque passage dans la boucle intérieure : il est donc de complexité $O(n \log_2 n)$.

Proc FAISCCONV(i,*candidates*,S)
  $Y \leftarrow$ *candidates* en ordre décroissant des dates échues;
  $somme\_p \leftarrow \sum_{k \in candidates} p_k$; $S \leftarrow \emptyset$;
  $trou \leftarrow$ **vrai**;
  TantQue $S = \emptyset$ **et** $|Y| \geq 1$ **et** $trou$ faire
    Si $d_{Y[1]} - r_i < somme\_p + p_i$ alors $S \leftarrow Y$; ①
    Sinon
      $trou \leftarrow$ **faux**;
      TantQue **non** $trou$ **et** $|Y| \geq 2$ faire ②
        Si $d_{Y[2]} \geq d_{Y[1]} - p_{Y[1]}$ alors
          $somme\_p \leftarrow somme\_p - p_{Y[1]}$;
          $Y \leftarrow Y \setminus \{Y[1]\}$;
        Sinon $trou \leftarrow$ **vrai**; FinSi
      FinTantQue
    FinSi
  FinTantQue
FinProc

Figure 3. – Algorithme de FAISCCONV.

3.3.2. *Procédure VÉRIFCONJ&ACTU_R*

La Procédure VÉRIFCONJ&ACTU_R (Fig. 4) s'appuie sur la remarque suivante pour actualiser au mieux les dates limites et les rangs.

Remarque : Soient $S_1 \subset T$, $S_2 \subset S_1$, deux faisceaux convergents pour $i \in T \setminus S_1$. D'après les expressions (5), les actualisations de $R_g(i)$ et r_i apportées par $i \not\prec S_2$ sont au moins aussi importantes que celles apportées par $i \not\prec S_1$. A l'inverse, si les relations conjonctives $S_1 \prec i$ et $S_2 \prec i$ sont vérifiées, d'après les expressions (3), les actualisations déduites de $S_1 \prec i$ sont au moins aussi importantes que celles déduites de $S_2 \prec i$.

Le faisceau S est ainsi réduit (boucles ① et ②) jusqu'à ce qu'une relation conjonctive $S \prec i$ ait été mise en évidence ($conj$ = **vrai**) ou qu'il n'y ait plus de réduction possible ($reduction$ = **faux**). Les dates limites sont ensuite actualisées par les instructions ③. La première phase de réduction (boucle ①) est basée sur la proposition 6.

PROPOSITION 6 : *Soient $S \subset T$, $|S| > 1$ et $i \in T \setminus S$ tels que S soit un faisceau convergent pour i. Soit Y la liste des tâches de S classées selon l'ordre décroissant de leur date de fin au plus tard. Si*

$$d_{Y[1]} - p_{Y[1]} \geq d_{Y[2]}, \tag{6}$$

alors $S \setminus \{Y[1]\}$ est également un faisceau convergent pour i.

Preuve : Comme S est un faisceau convergent pour i,

$$\max_{s \in S} d_s - r_i < \sum_{s \in S} p_s + p_i,$$

ou encore, $d_{Y[1]} - r_i < \sum_{s \in S \setminus \{Y[1]\}} p_s + p_{Y[1]} + p_i$, *i.e.*,

$$d_{Y[1]} - p_{Y[1]} - r_i < \sum_{s \in S \setminus \{Y[1]\}} p_s + p_i.$$

Avec l'expression (6), on obtient $d_{Y[2]} - r_i < \sum_{s \in S \setminus \{Y[1]\}} p_s + p_i$, ou encore,

$$\max_{s \in S \setminus \{Y[1]\}} d_s - r_i < \sum_{s \in S \setminus \{Y[1]\}} p_s + p_i.$$

$S \setminus Y[1]$ est un faisceau convergent pour j. □

La seconde phase de réduction (②) supprime des tâches dans l'ordre croissant des dates de fin au plus tôt ($r_s + p_s$, $s \in S$). Cet ordre de suppression a été choisi pour favoriser l'augmentation de la plus petite date de fin au plus tôt des tâches de S, qui conditionnera l'actualisation de r_i, i telle que $i \not\prec S$ (*cf.* expressions (5)).

La complexité de la procédure VÉRIFCONJ&ACTU_R est celle du tri de Z qui peut être effectué en $O(n \log_2 n)$. Il est à noter que dans un souci de simplification de la procédure, l'étape de réduction n'est pas optimale ; elle ne garantit en effet, ni l'obtention de l'ensemble S maximal tel que $S \prec i$, ni celle du faisceau minimal $i \not\prec S$.

3.3.3. *Discussion*

Dans la littérature [4, 5, 16], les complexités des procédures liées à l'application des règles de précédences conjonctives correspondent à *une* application des règles. Ainsi, Carlier et Pinson (1990) et Nuijten (1994) proposent un algorithme en $O(n^2)$, Carlier et Pinson (1994) en $O(n \log_2 n)$ pour les mises à jour de toutes les tâches. Lorsqu'on applique des règles de précédences non conjonctives, le résultat de la propagation peut être différent selon l'ordre où les règles ont été appliquées. Cette caractéristique brise la sémantique de *point « fixe »* qui assure que le résultat final de la propagation ne dépend pas de l'ordre de ses étapes. La recherche du point fixe global amène à exploiter au maximum les règles et donc appeler plusieurs fois chacun des algorithmes. Dans notre étude, cette phase est d'autant plus importante que d'autres contraintes que les contraintes de dates sont propagées ; les rangs peuvent ainsi être actualisés, et donc contribuer à la relance de l'algorithme, alors que les dates n'ont pas subi de modification. En toute rigueur, si l'on tient compte de cette caractéristique, les algorithmes mentionnés ci-dessus sont à considérer avec des complexités de $O(\lambda n^3)$ pour [4] et [16], $O(\lambda n^2 \log_2 n)$ pour [5].

La procédure PRÉCENS que nous proposons ici est en $O(\lambda n^3 \log_2 n)$, précédences conjonctives et non conjonctives comprises. Comparée à [16], cette performance paraît tout à fait compétitive puisque Nuijten propose une procédure en $O(\lambda n^4)$ pour les précédences non conjonctives – associée à l'algorithme SEQUENCINGCHECK2 en $O(n^3)$ dans [17]. Il semble en revanche qu'elle soit moins intéressante que celle de Carlier et Pinson (1994), mais en contrepartie elle utilise des structures de données moins complexes et permettant des déductions plus importantes – au niveau des rangs notamment – de par la recherche des faisceaux non conjonctifs actualisants.

Notons enfin que des travaux récents ont permis des progrès importants dans la recherche et le traitement des précédences non conjonctives. Baptiste et Le Pape proposent ainsi un algorithme en $O(n^2)$ assurant la complétude des actualisations trouvées [2]. Cette complexité correspond toutefois à l'application des règles sur une seule passe. La procédure doit donc être exécutée de façon répétitive jusqu'à ce que les domaines des variables deviennent stables. Dans le pire des cas, la complexité passe donc en $O(\lambda n^3)$ même si peu d'exécutions consécutives interviennent en pratique. Dans [20] est proposé un développement de PRÉCENS, toujours en $O(\lambda n^3 \log_2 n)$. Une seule passe suffit pour trouver toutes les actualisations ; en outre, la condition de détection des faisceaux est plus forte que dans [2]. Il reste toutefois à

mener des tests comparatifs pour juger de l'efficacité respective des deux algorithmes.

Proc VÉRIFCONJ&ACTU_R(i,S)
{S a été rangé par FAISCCONV en ordre décroissant des dates échues}
  $somme\_p \leftarrow \sum_{s\in S} p_s$; $reduction \leftarrow$ **vrai**; $conj \leftarrow$ **faux**;
  TantQue $reduction$ **et** **non** $conj$ **et** $|S| \geq 2$ faire ①
    Si $d_{S[1]} - \min_{s\in S} r_s < somme\_p + p_i$ alors $conj \leftarrow$ **vrai**; $\{S \prec i\}$ (*)
    Sinon
      Si $d_{S[2]} \leq d_{S[1]} - p_{S[1]}$ alors
        {$S \setminus \{S[1]\}$ est toujours un faisceau convergent pour i}
        $somme\_p \leftarrow somme\_p - p_{S[1]}$; $S \leftarrow S \setminus \{S[1]\}$;
      Sinon $reduction \leftarrow$ **faux**; FinSi
  FinTantQue
  $reduction \leftarrow$ **vrai**;
  $Z \leftarrow S \setminus \{S[1]\}$ en ordre croissant des dates de fin au plus tôt;
  TantQue $reduction$ **et** **non** $conj$ faire ②
    Si $d_{S[1]} - \min_{s\in S} r_s < somme\_p + p_i$ alors $conj \leftarrow$ **vrai**; $\{S \prec i\}$ (*)
    Sinon
      Si $Z \neq \emptyset$ alors
        Si $d_{S[1]} - r_i < somme\_p - p_{Z[1]} + p_i$ alors
          {$S \setminus \{Z[1]\}$ est toujours un faisceau convergent pour i}
          $somme\_p = somme\_p - p_{Z[1]}$; $Z \leftarrow Z \setminus \{Z[1]\}$; $S \leftarrow \{S[1]\} \cup Z$;
        Sinon $reduction \leftarrow$ **faux**; FinSi
      Sinon $reduction \leftarrow$ **faux**; FinSi
    FinSi
  FinTantQue
  Si $conj$ alors {précédences conjonctives : $S \prec i$} ③
    $r_i \leftarrow \max[r_i, eft(S)]$; $R_g(i) \leftarrow \max[R_g(i), llr(S) + 1]$;
    Si $d_i - r_i < p_i$ alors arrêt; FinSi {problème incohérent}
    PourTout $s \in S$ faire
    PourTout $s \in S$ faire
      $d_s \leftarrow \min(d_s, d_i - p_i)$; $R_d(s) \leftarrow \min[R_d(s), R_d(i) - 1]$;
      Si $d_s - r_s < p_s$ alors arrêt; FinSi {problème incohérent}

FinPourTout
Sinon {précédences non conjonctives : $i \not\prec S$}
$r_i \leftarrow \max[r_i, \min_{s \in S}(r_s + p_s)]$; $R_g(i) \leftarrow \max[R_g(i), \min_{s \in S} R_g(s) + 1]$;
Si $d_i - r_i < p_i$ alors arrêt; FinSi {problème incohérent}
FinSi
FinProc

(*) Comme S est un faisceau convergent pour i, on a $i \not\prec S$ et il n'est pas nécessaire de considérer le cas où la plus petite date de début au plus tôt est r_i dans la proposition 3.

Figure 4. – Algorithme de VÉRIFCONJ&ACTU_R.

3.4. Exemple

L'exemple 1 présenté en 2.3 est de nouveau considéré. À partir des données actualisées par PRÉCPAIRES, PRÉCENS permet de déduire les relations $1 \prec \{2,3\}$ et $\{4,5,7\} \not\prec 6$, puis $6 \prec 7$. Elles conduisent aux actualisations indiquées en gras dans le tableau 2a. Une nouvelle itération de la procédure PRÉCPAIRES déduit les actualisations du tableau 2b par les relations $Prec(7) = \{1,2,3,4,5,6\}$ et $Suiv(5) = \{7\}$.

TABLEAU 2

Ex. 1 : Données actualisées par PRÉCPAIRES, puis PRÉCENS (a) et de nouveau PRÉCPAIRES (b).

i	1	2	3	4	5	6	7
r_{ai}	0	**2**	**2**	4	4	7	**10**
d_{ai}	**2**	4	4	8	11	**11**	13
$R_{g_a}(i)$	1	**2**	**2**	4	4	5	**6**
$R_{d_a}(i)$	**1**	3	3	5	7	**6**	7

(*a*)

i	1	2	3	4	5	6	7
r_{ai}	0	2	2	4	4	7	**11**
d_{ai}	2	4	4	8	11	11	13
$R_{g_a}(i)$	1	2	2	4	4	5	**7**
$R_{d_a}(i)$	1	3	3	5	**6**	6	7

(*b*)

4. DÉCOMPOSITION

4.1. Structure décomposée proposée

Dans [14, 15], nous avons défini trois niveaux de regroupement des tâches : les Groupes Autonomes de Tâches (GAT), les Groupes de Tâches à Rangs Egaux (GTRE) et les Groupes de Tâches à Rangs Inclus (GTRI).

Les GAT ([1]) sont indépendants au sens du séquencement mais sont souvent de taille trop importante pour être faciles à manipuler. Les tâches des GTRE sont permutables vis-à-vis des conditions d'admissibilité mises en évidence; l'ordonnancement d'un GTRE est de ce fait facile à aborder, mais pour de nombreux problèmes, les GTRE sont trop couplés. Une configuration idéale, conduisant à des décompositions en GAT et en GTRE identiques, permettrait d'obtenir des sous-problèmes indépendants et faciles à résoudre. Dans le cas général, le niveau intermédiaire des GTRI peut répondre à un compromis entre l'indépendance des groupes et la permutabilité des tâches au sein d'un groupe. C'est ce niveau de décomposition qui a été choisi.

DÉFINITION 1 : *Soient $i \in T$, $j \in T$, $i \neq j$. Les tâches i et j appartiennent au même GTRI I si et seulement si :*

$$\mathcal{R}(i) \subseteq \mathcal{R}(j) \text{ ou } \mathcal{R}(j) \subseteq \mathcal{R}(i) \text{ ou } \begin{cases} \exists k \in I, \ k \neq i, j, \text{ tel que :} \\ \mathcal{R}(i) \subset \mathcal{R}(k) \text{ et } \mathcal{R}(j) \subset \mathcal{R}(k). \end{cases}$$

i et j ont des intervalles de rangs inclus ou il existe une tâche $k \in I$ dont l'intervalle de rangs inclut $\mathcal{R}(i)$ et $\mathcal{R}(j)$.

DÉFINITION 2 : *L'intervalle de rangs $\mathcal{R}(I)$ d'un GTRI I est la réunion des intervalles de rangs des tâches de I :*

$$\mathcal{R}(I) = \bigcup_{i \in I} \mathcal{R}(i) = \left[\min_{i \in I} R_g(i), \max_{i \in I} R_d(i) \right].$$

4.2. Algorithme de décomposition

La procédure DÉCOMPGTRI construit une partition en GTRI selon l'algorithme de la figure 5. Elle examine séquentiellement les tâches classées en ordre croissant de leur rang à gauche (dans le tableau X) et décide,

([1]) Les « sous-séquences semi-rigides » introduites dans [1] sont les GAT et les regroupements de GAT.

pour chacune d'elles, d'une affectation au GTRI courant ou de la création d'un nouveau GTRI.

La décomposition en GTRI n'est pas unique. Parmi les différentes partitions possibles, notre procédure privilégie celle qui affecte les tâches pouvant appartenir à deux GTRI à celui dont le nombre de rangs admissibles est le plus petit. Dans les cas – rares – d'appartenance possible à plus de deux GTRI, seuls les deux premiers GTRI rencontrés au cours du déroulement de l'algorithme sont considérés. La gestion de l'affectation des tâches susceptibles d'appartenir à deux GTRI n'est ici qu'évoquée. L'algorithme complet expliquant cet aspect de la procédure est décrit dans [15]. Sa complexité est celle du tri de T pour la construction de la liste X qui peut être effectué en $O(n \log_2 n)$.

Proc DÉCOMPGTRI(T,$\mathcal{I}$) {$\mathcal{I}$ représente l'ensemble des GTRI}
  $X \leftarrow T$ en ordre croissant des rangs à gauche;
  $\mathcal{I} \leftarrow \emptyset$; $I \leftarrow \{X[1]\}$;
  Pour x de 2 à n faire
    Si $\mathcal{R}(X[x]) \subseteq \mathcal{R}(I)$ ou $\mathcal{R}(I) \subset \mathcal{R}(X[x])$ alors {$X[x]$ peut appartenir à I}
      $I \leftarrow I \cup \{X[x]\}$;
      mémoriser les tâches qui peuvent appartenir au GTRI précédemment créé;
      mémoriser les tâches susceptibles d'appartenir au prochain GTRI;
    Sinon {un nouveau GTRI doit être créé pour $X[x]$}
      affecter les tâches pouvant appartenir aux deux derniers GTRI;
      $\mathcal{I} \leftarrow \mathcal{I} \cup I$; $I \leftarrow \{X[x]\}$;
    FinSi
  FinPour
  $\mathcal{I} \leftarrow \mathcal{I} \cup I$;
FinProc

Figure 5. – Algorithme de DÉCOMPGTRI.

4.3. Exemple

L'exemple 1 dont les dates limites et les rangs ont été précédemment actualisés par les procédures PRÉCPaires et PRÉCEns (*cf.* §2.3 et §3.4) est maintenant décomposé en GTRI. Les GTRI de cet exemple sont dessinés sur la figure 6 où les rangs représentés sont ceux du tableau 2b. Pour cet exemple particulier, les GTRI sont aussi des GAT.

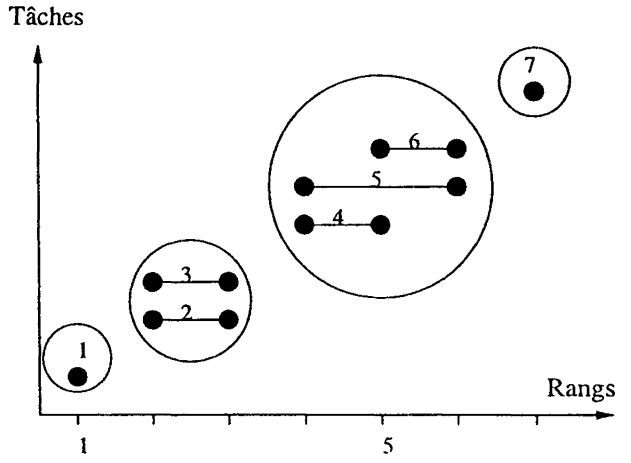

Figure 6. – Ex. 1 : décomposition en GTRI.

5. ÉVALUATION DE L'APPROCHE PROPOSÉE

Notre approche de décomposition par caractérisation de solutions admissibles fournit, d'une part une structuration du problème en GTRI et d'autre part des dates limites et des rangs actualisés. La stratégie adoptée pour l'appel des différentes procédures est décrite figure 7 (procédure CARAC&DÉCOMP) ; les précédences entre une tâche et un ensemble de tâches ne sont recherchées que sur les sous-problèmes que constituent les GTRI, de manière à réduire le temps de calcul requis pour caractériser les solutions admissibles.

Proc CARAC&DÉCOMP(T,$\mathcal{I}$)
  Répéter
    PRÉCPAIRES(T);
    Si une incohérence a été détectée alors arrêt; FinSi
    DÉCOMPGTRI(T,$\mathcal{I}$);
    PourTout $I \in \mathcal{I}$ faire
      PRÉCENS(I);
      Si une incohérence a été détectée alors arrêt; FinSi
    FinPourTout
  Jusqu'à ce qu'il n'y ait plus d'actualisation
FinProc

Figure 7. – Algorithme de CARAC&DÉCOMP.

Nous évaluons maintenant dans quelle mesure notre approche permet de préparer la phase de recherche d'une solution, qu'elle soit menée par un opérateur humain ou par une procédure de résolution automatique.

5.1. Aide à la décision d'un opérateur humain

Dans un contexte d'aide à la décision, la structure décomposée proposée offre des perspectives pour guider un opérateur humain dans ses choix. Une aide est fournie à l'utilisateur en lui proposant des ensembles de tâches de taille réduite à ordonnancer. Alors que dans une méthode de classification automatique, il est possible de choisir le nombre de groupes de la partition ou le cardinal maximal d'un groupe, le choix d'un niveau d'agrégation (GTRI) est ici guidé par les propriétés des groupes. La transmission des dates limites et des rangs actualisés permet à l'opérateur d'approfondir sa connaissance du problème en disposant d'informations temporelles et séquentielles complémentaires. Par exemple, lorsque les GTRI obtenus par la décomposition ne sont pas des GAT – groupes indépendants au sens du séquencement –, les conflits qui résultent du partage de certains rangs par plusieurs GTRI doivent être résolus. La visualisation des intervalles de rangs des tâches peut aider un opérateur humain à résoudre ces conflits, selon des critères qui lui sont propres et liés au contexte de fabrication, souvent motivés par des aléas (perturbations, modifications dans les demandes de production, commandes urgentes, *etc.*).

Notre approche a permis d'enrichir les conditions d'admissibilité mises en évidence par Amamou *et al.* [1] en ajoutant au calcul des marges temporelles une recherche de relations séquentielles entre une tâche et un ensemble de tâches et en procédant à des actualisations de dates limites et de rangs. La définition des GTRI a de plus conduit à affiner la structure décomposée proposée par cette méthode. Plus récemment, Happiette et Zeng [11] ont ajouté à l'approche d'Amamou *et al.* une procédure d'énumération partielle des solutions, illustrée sur un problème à une machine de sept tâches et sur des flow-shops. Le problème à une machine a été décomposé par notre approche ; elle met en évidence sept GAT et montre donc d'emblée qu'une seule séquence est admissible. La procédure de décomposition de Happiette et Zeng ne détermine que trois GAT ; la construction d'un arbre d'énumération partielle des solutions comprenant trente-trois sommets est ensuite nécessaire pour montrer qu'une seule séquence est admissible.

Des expérimentations plus systématiques ont été menées pour quantifier le nombre moyen d'actualisations de dates et de rangs, et la réduction de fenêtre

temporelle ou d'intervalle de rangs qu'elles représentent. Les problèmes testés ont été générés aléatoirement selon des lois uniformes sur les intervalles de variations indiqués dans le tableau 3. Les résultats expérimentaux reportés dans le tableau 4 sont des moyennes sur 200 instances admissibles ([2]) générées :

- du nombre de fenêtres temporelles actualisées, *actu_dates*,
- de la réduction moyenne de marge temporelle apportée par chaque actualisation,

$$reduc\_marge = \frac{1}{actu\_dates} \sum_{i=1}^{n} \left(\frac{(d_i - r_i - p_i) - (d_{a_i} - r_{a_i} - p_i)}{d_i - r_i - p_i} \right),$$

- du nombre d'intervalles de rangs actualisés, *actu_rangs*,
- de la réduction moyenne d'intervalle de rangs apportée par chaque actualisation,

$$reduc\_rangs = \frac{1}{actu\_rangs} \sum_{i=1}^{n} \left(\frac{[R_d(i) - R_g(i)] - [R_{d_a}(i) - R_{g_a}(i)]}{R_d(i) - R_g(i)} \right),$$

- du temps cpu consommé par la phase de caractérisation et correspondant à une station de travail SUN-SPARC5 32MB et à une implémentation des procédures en langage C.

TABLEAU 3

Séries de problèmes générés.

$r_i \in [0, r_{max}]$, $p_i \in [1, 200]$, $d_i \in [r_i + p_i, r_i + p_i + a_{max}]$, $\forall i = 1...n$							
Série	1	2	3	4	5	6	7
r_{max}	50000	50000	50000	50000	50000	50000	150000
a_{max}	50000	40000	30000	20000	10000	5000	50000
n	200	200	200	200	200	200	1000

Pour les séries 1 à 6, le nombre d'actualisations de dates et de rangs est lié aux variations de a_{max}. Quand a_{max} décroît, le problème est plus contraint, si bien que les déductions ont des chances d'être plus nombreuses, d'où un nombre plus important d'actualisations et une plus grande réduction des marges temporelles et des intervalles de rangs. Il en résulte, par l'activation des conditions de boucles « Répéter ... Jusqu'à ce qu'il n'y ait plus d'actualisation », une augmentation du temps cpu consommé. Les résultats de la série 7 sont donnés pour illustrer la caractérisation de problèmes de 1000 tâches.

([2]) Les problèmes qui ne possèdent pas de solution admissible ne sont pas considérés ; ils sont identifiés par une procédure optimale de minimisation du plus grand retard algébrique.

TABLEAU 4
Pouvoir actualisant de CARAC&DÉCOMP.

Série	1	2	3	4	5	6	7
actu_dates	0,54	0,665	0,995	1,535	3,465	6,79	4,8
reduc_marge	0,90%	1,22%	2,41%	2,05%	3,57%	6,20%	1,26%
actu_rangs	198,9	200	200	200	200	200	1000
reduc_rangs	33,64%	41,52%	52,32%	65,70%	81,75%	90,73%	70,56%
Temps cpu (s)	0,87s	1,13s	1,50s	2,05s	2,97s	3,73s	242,32s

Les actualisations de dates peuvent paratre faibles, aussi bien en nombre qu'en pouvoir de réduction de fenêtre temporelle. Néanmoins, le paragraphe suivant montrera qu'aussi peu nombreuses soient-elles, elles sont souvent déterminantes puisque la caractérisation des solutions admissibles permet d'améliorer de façon significative les résultats fournis par des heuristiques classiques pour la minimisation du plus grand retard algébrique.

Des tests ont également été effectués avec PRÉCENS appliquée au problème non décomposé (Tab. 5).

TABLEAU 5
PRÉCENS appliquée au problème global.

PRÉCENS sur le problème global							
	Série 1	Série 2	Série 3	Série 4	Série 5	Série 6	Série 7
actu_dates	0,54	0,665	0,995	1,535	3,47	6,805	4,805
reduc_marge	0,90 %	1,22 %	2,41 %	2,05 %	3,57 %	6,20 %	1,25 %
actu_rangs	198,90	200	200	200	200	200	1000
reduc_rangs	33,64 %	41,52 %	52,32 %	65,70 %	81,75 %	90,73 %	70,56 %
Temps cpu	2,83s	2,94s	3,15s	3,52s	4,33s	5,6s	443,5s

N.B. Les valeurs décimales de *actu_dates* et *actu_rangs* sont exactes, tandis que les valeurs de *reduc_marge* et *reduc_rangs* ont été arrondies.

Il apparaît peu de conditions d'admissibilité supplémentaires : on note sur 200 instances générées, une actualisation de fenêtre temporelle de plus pour les séries 5 et 7 et trois pour la série 6. La très faible variation de *reduc_marge* montre que ces actualisations supplémentaires entraînent une réduction de fenêtre temporelle équivalente à celle des autres actualisations. Le gain de temps cpu engendré par la décomposition en GTRI représente environ 50 % du temps cpu requis pour l'appel de la procédure PRÉCENS sur le problème global. Ce gain nous a paru justifier la stratégie de recherche

des relations séquentielles entre une tâche et un ensemble de tâches sur chaque GTRI uniquement.

5.2. Résolution par une procédure automatique

Avant tout destinée à l'aide à la décision d'un opérateur humain, la caractérisation des solutions admissibles a néanmoins été étudiée dans le cadre de la génération de solutions afin de concrétiser l'exploitation possible des conditions d'admissibilité.

Une solution approchée au problème de minimisation du plus grand retard algébrique ($L_{\max}$) est ainsi déterminée. Rappelons que le problème $(n/1/r_i, d_i/L_{\max})$ a été démontré NP-difficile au sens fort [13]. Il est néanmoins bien résolu par la procédure arborescente de Carlier [3] qui permet de déterminer rapidement la solution optimale des problèmes comportant jusqu'à un millier de tâches. Notre approche ne se justifierait donc pas si elle avait pour seule vocation de minimiser le $L_{\max}$; nous avons déjà souligné l'exploitation possible des informations qu'elle fournit dans une optique d'aide à la décision d'un opérateur humain. Ceci explique sans doute pourquoi davantage d'efforts n'ont pas été consacrés à l'amélioration des calculs des bornes des solutions optimales, à l'aide en particulier d'une procédure par séparations et évaluations. De même, l'optimisation d'autres critères liés au retard pourrait également être envisagée. Notre but principal est simplement d'évaluer la possibilité d'utiliser notre approche comme phase préparatoire à la résolution d'un problème d'optimisation.

La décomposition en GTRI aurait pu être exploitée en procédant à une résolution optimale locale (par l'algorithme de Carlier) des sous-problèmes que constituent les GTRI. Cette voie n'a pas été suivie car il nous a semblé que les couplages importants qui apparaissent généralement entre les GTRI nécessiteraient l'utilisation d'une procédure de coordination des solutions locales trop sophistiquée. L'utilisation des intervalles de rangs pour la génération d'une solution approchée à la minimisation du $L_{\max}$ a été envisagée [14, 15] ; la procédure HEURISTRANGS développée consiste à modifier la solution de Jackson pour qu'elle respecte au mieux les intervalles de rangs. Les expérimentations ont toutefois montré que l'ordonnancement actif associé à la règle EDD était généralement plus proche de l'optimum que la solution de Jackson. La mise en œuvre d'une procédure similaire à HEURISTRANGS basée cette fois sur la génération d'ordonnancements actifs devrait de ce fait être entreprise.

Dans le cadre de ce paragraphe, nous nous intéressons seulement à l'exploitation possible des conditions d'admissibilité temporelles mises en évidence à l'issue de la procédure CARAC&DÉCOMP (dates limites actualisées).

Deux heuristiques classiques ont été appliquées aux problèmes des séries 1 à 7 (Tab. 3), d'une part avec les dates limites initiales et d'autre part avec les dates limites actualisées. Les résultats de ces expérimentations sont données dans le tableau 6 ; ils concernent respectivement l'algorithme de Jackson [12] qui génère un ordonnancement sans délai avec la règle de priorité EDD (Earliest Due Date) et l'algorithme de génération d'un ordonnancement actif avec cette même règle EDD [18].

Dans ce tableau, $\%L$ et *%nb_opt* représentent les moyennes sur 200 instances admissibles de l'écart relatif à l'optimum et du pourcentage de solutions optimales atteintes, l'optimum étant chaque fois calculé par l'algorithme de Carlier.

TABLEAU 6

Algorithmes classiques appliqués aux données initiales et aux données actualisées.

Série	OSD1		OSD2		OAC1		OAC2	
	%nb_opt	$\%L$	*%nb_opt*	$\%L$	*%nb_opt*	$\%L$	*%nb_opt*	$\%L$
1	61	26,5	66,5	5	87,5	19,5	89,5	2,5
2	57	34	63	7	88	15	90	3,5
3	55,5	49,5	63,5	7,5	86,5	26,5	88,5	4,5
4	52,5	93,5	63,5	17	85	40,5	89,5	5,5
5	40	314,5	60,5	96	82,5	84	88	35
6	27	563,5	55,5	117	78,5	111	88,5	17,5
7	25,5	618,5	47	286,5	57	307	64	127,5

OSD : Ordonnancement Sans Délai.
OAC : Ordonnancement ACtif.
1 : Avant caractérisation (dates limites initiales).
2 : Après caractérisation (dates limites actualisées).
%nb_opt : Pourcentage de solutions optimales atteintes.
$\%L$: Écart entre la valeur obtenue pour le $L_{\max}$ et la valeur de l'optimum en pourcentage par rapport à l'optimum.

Il apparaît une amélioration significative des résultats lorsque les deux heuristiques sont appliquées aux données actualisées. Les temps cpu restent du même ordre de grandeur et sont dans tous les cas inférieurs à la seconde. Notons que pour le type de problèmes testés et le critère $\min L_{\max}$, la génération d'ordonnancements actifs fournit en moyenne des solutions bien plus proches de l'optimum que la génération d'ordonnancements sans délai.

6. CONCLUSION

Nous avons présenté une approche par décomposition temporelle basée sur la détermination d'intervalles de rangs admissibles ; elle a pour objectif de préparer la phase de génération de solutions. Les résultats expérimentaux ont permis de quantifier le pouvoir actualisant des procédures proposées. Ils ont de plus montré que l'actualisation des dates limites permettait d'améliorer de façon significative les résultats fournis par les algorithmes de génération d'ordonnancements actifs ou sans délai avec la règle EDD.

Au vu des résultats obtenus à ce jour, plusieurs pistes peuvent être dégagées. Certaines améliorations pourraient être apportées à l'algorithme de recherche de précédences entre une tâche et un ensemble de tâches. La recherche de tous les faisceaux *minimaux* actualisants pourrait notamment être entreprise (actuellement, chaque faisceau actualisant trouvé est réduit de façon partielle) ; ou mieux encore : utiliser les travaux récents permettant de mettre en place un algorithme efficace assurant la complétude des actualisations. Il serait alors profitable de procéder à des tests comparatifs avec les algorithmes de recherche exhaustive de précédences conjonctives et non conjonctives développés par Wim Nuijten.

Une procédure exploitant la structuration en GTRI et basée sur la génération d'ordonnancements actifs permettrait sans doute d'améliorer les résultats dans la recherche de solutions au problème de minimisation du plus grand retard algébrique.

En ce qui concerne l'aide à la décision d'un opérateur humain, il reste à évaluer la pertinence des informations fournies. Dans une optique à plus long terme, il serait intéressant de développer un système interactif d'aide à la décision, proposant à un opérateur humain la structure décomposée en la faisant évoluer à chaque prise de décision.

RÉFÉRENCES

1. M. Amamou, M. Happiette et M. Staroswiecki, *Decomposition of the single machine scheduling based on the notion of semi-rigid subsequences*, Proc. International Conference on Automation Technology, Taiwan, 4-6 Juillet 1992.
2. P. Baptiste et C. Le Pape, *Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling*, Proc. of U.K. planning and scheduling, SIG meeting, Liverpool, Novembre 1996.
3. J. Carlier, The one-machine sequencing problem, *European Journal of Operational Research*, 1982, *11*, p. 42-47.
4. J. Carlier et E. Pinson, A practical use of Jackson's preemptive schedule for solving the job-shop problem, *Annals of Operations Research*, 1990, *26*, 269-287.

5. J. CARLIER et E. PINSON, Adjustment of heads and tails for the job-shop problem, *European Journal of Operational Research*, 1994, *78*, p. 146-161.
6. R. CHAMBERS, R. CARRAWAY, T. TOWE et T. MORIN, Dominance and decomposition heuristics for single machine scheduling, *Operations Research*, 1991, *39*, p. 639-647.
7. J. ERSCHLER, F. ROUBELLAT et J.-P. VERNHES, Characterizing the set of feasible sequences for n jobs to be carried out on a single machine, *European Journal of Operational Research*, 1980, *4*, p. 189-194.
8. J. ERSCHLER, G. FONTAN et C. MERCÉ, Approche par contraintes en planification et ordonnancement de la production, *APII*, 1993, *27*, p. 669-695.
9. P. ESQUIROL, *Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de contraintes*, Thèse de doctorat de l'Université Paul Sabatier, Toulouse, 1987, Rapport LAAS No 87291.
10. GOThA, Les problèmes d'ordonnancement, *RAIRO Rech. Opér.*, 1993, *27*, n° 1, p. 77-150.
11. M. HAPPIETTE et X. ZENG, Ordonnancement pour le problème du flow-shop, *APII*, 1995, *29*, p. 623-639.
12. J. R. JACKSON, *Scheduling a production line to minimize maximum tardiness*, Research report 43, Management research project, University of California, Los Angeles, CA, 1955.
13. J. K. LENSTRA, A. H. G. RINNOOY KAN et P. BRÜCKER, Complexity of machine scheduling problems, *Annals of discrete mathematics*, 1977, *1*, p. 343-362.
14. M.-L. LEVY, P. LOPEZ et B. PRADIN, *A decomposition approach for the single-machine scheduling problem*, Journal of Decision Systems, *5*, n° 1-2, p. 73-94, 1996.
15. M.-L. LEVY, *Décomposition temporelle et problèmes d'ordonnancement*, Thèse de doctorat de l'Institut National Polytechnique de Toulouse, Mars 1996, Rapport LAAS No 96079.
16. W. P. M. NUIJTEN, *Time and Resource Constrained Scheduling*, PhD thesis, Eindhoven University of Technology, The Netherlands, 1994.
17. W. P. M. NUIJTEN et E. H. L. AARTS, A computational study of constraint satisfaction for multiple capacitated job-shop scheduling, *European Journal of Operational Research*, 1996, *90*, p. 269-284.
18. M. PINEDO, *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs, 1995.
19. M.-C. PORTMANN, Méthodes de décomposition spatiale et temporelle en ordonnancement de la production, *APII*, 1988, *22*, p. 439-451.
20. P. TORRES, *Techniques de propagation de contraintes et problèmes d'ordonnancement*, Mémoire de DEA, Institut National des Sciences Appliquées, Toulouse, Septembre 1996.