Contrôle de cohérence et établissement d'un ordre d'exécution dans un problème d'ordonnancement

Dr Malika BABES

Université BADJI Mokhtar
Faculté des Scienes - Département d'informatique
BP 12, 23000 Annaba - ALGERIE
Tel et Fax : (213) 08 87 27 56

Résumé : *Nous considérons ici un ensemble de processus soumis à une variété de contraintes, entre autres, la contrainte de disjonction. Chaque processus est muni d'une fenêtre de temps dans laquelle il est sensé être exécuté.*
Afin de vérifier la cohérence des contraintes entre elles, nous définissons un ensemble de règles d'inférence. Leur propagation permet de déterminer un ensemble de solutions réalisables. A base de cet ensemble, nous établissons un ordonnancement qui privilégie à chaque fois le processus le plus urgent.

Mots clés : *Ordonnancement, Date de disponibilité, Date échue, Disjonction, Règle d'inférence, Propagation de contraintes.*

Abstract : *We deal here with a set of tasks which are submitted to somme constraints like a disjonction constraint. Each task has a release time and a due date.*
Our objective, in the first time, is to check the coherence of all the constraints. So, we define a set of inference rules. Their propagation allows us to designate a set of admissible solutions. From this set, we define, in the second time, a schedule which favours, each step, the process with the least due date.

Key words : *Scheduling, Release time, Due date, Disjonction, Inference rule, Constraints propagation.*

1. Introduction :

Nous considérons dans cette étude un ensemble X de processus de longueurs fixes. Les contraintes qui les relient correspondent à des relations binaires et unaires de positionnements relatifs entre ces processus. On distingue en particulier :

- *des contraintes binaires telles que : avant, commence-comme, finit-comme, commence-quand-finit, inclut et disjoint-de,*
- *des contraintes unaires telles que : $disponible(x) \geq date$ et $échue(x) \leq date$,*

où date est une constante donnée pour chaque $x \in X$. Par $disponible(x)$ et $échue(x)$, $\forall x \in X$, nous désignons, respectivement, la première date à laquelle le processus x est disponible pour un traitement, et la dernière date à laquelle x est sensé être terminé. Imposer de telles dates revient tout simplement à exécuter chaque processus dans une fenêtre de temps prédéfinie, probablement pour des raisons technologiques.

Nous sommes appelés, en première phase, à vérifier la cohérence ou le caractère non antagoniste des contraintes entre elles. En seconde phase, nous devons dater les processus tout en respectant les contraintes auxquelles ils sont soumis.

Dans tout ce qui va suivre, et dans un but de simplification d'écriture, nous notons, respectivement, par P_x, r_x, d_x, C_x et F_x, la durée, la date de disponibilité, la date échue, le commencement et la fin de la tâche $x \in X$.

Signalons au début que le séquencement de tâches de durées quelconques et dont les dates de disponibilité et échues sont données, est un problème NP-difficile au sens fort [LEN 76]. De nombreux chercheurs ont essayé d'obtenir des algorithmes polynomiaux pour résoudre le problème dans le cas où les durées des tâches sont égales [FLO 71, LAG 76, CAR 84]. Leur approches sont toutes basées sur la règle de Jackson [JAC 55] qui consiste à prendre en charge, en premier, la tâche la plus urgente. Néanmoins, il existe des algorithmes polynomiaux dans le cas où la preemption est permise [MUN 70, BAK 74, CAR 84, SAU 87, SAU 89].

2. Contrôle de cohérence des contraintes :

Les techniques de détection d'incohérence les plus fréquemment utilisées sont basées, soit sur une propagation de contraintes (approche de ALLEN) [ALL 81, ALL 84, RIT 88, GHA 89], soit sur la recherche d'un circuit absorbant dans un réseau [ROY 70, NIC 91].

2.1 Approche de ALLEN :

Dans son approche de détection d'incohérence d'un plan, ALLEN utilise treize relations de positionnement entre deux intervalles donnés, telles que le montre le tableau suivant. Ces treize relations s'excluent mutuellement :

relation	symbole de la relation	symbole de la relation inverse
before	b	a (after)
equals	=	
meets	m	mi
overlaps	o	oi
during	d	di
starts	s	si
finishes	f	fi

Tableau 1. *Représentation des treize relations de ALLEN*

A partir de ces relations, ALLEN construit des relations composées ; ce qui donne un réseau dont les sommets sont les intervalles correspondant aux durées des processus et où les arcs sont valués par la relation composée liant les deux intervalles.

Soit R l'ensemble des relations primitives ainsi définies. A tout couple $(r, r') \in R^2$ ALLEN fait correspondre un sous ensemble $T(r, r')$ de R, tel que :
si $x\ r\ y$ et $y\ r'\ z$, alors $T(r, r')$ est l'ensemble des relations possibles entre x et z.
La table $T(13 \times 13)$ entre les relations primitives permet facilement, par distributivité de la relation $\vee$, de calculer le résultat de n'importe quelle conjonction de deux relations composées. Le processus de résolution fonctionne alors grâce aux deux règles d'inférence suivantes :
Soit x, y et z trois processus, et Possible(x,y) l'ensemble des relations possibles entre x et y :

$$Poss := \bigvee_{r \in Possible(x,y)} \bigvee_{r' \in Possible(y,z)} T(r, r') \longrightarrow Possible(x, z) := Possible(x, z) \quad Poss$$

$$Possible(x, y) = \phi \longrightarrow \text{incohérence}$$

Ce schéma est largement inconsistant, puisqu'il ne peut détecter une situation telle que x, y et z de durées 1, deux à deux disjoints, et tous inclus dans a de durées 2.

2.2 Approche par circuit absorbant :

Elle consiste à construire un réseau dont les sommets sont les débuts et les fins des processus étudiés ; chaque arc est muni d'une longueur $l \in \mathbb{Z}$ exprimant une inégalité de potentiel. Un circuit absorbant (de longueur négative) est alors cherché (à l'aide d'une méthode classique, telle que la méthode de Dantzig). La détection d'un tel circuit implique l'incohérence des contraintes entre elles.

2.3 Proposition d'un ensemble de règles d'inférence pour résoudre le problème :

2.3.1 Présentation :

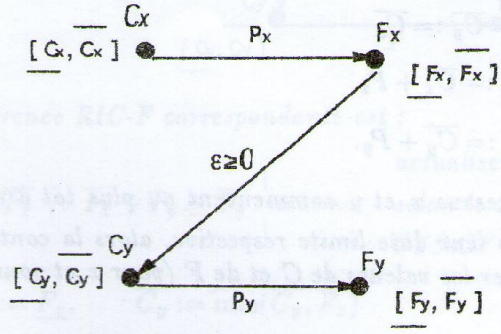
L'approche que nous proposons consiste à étudier la propagation d'un ensemble de règles d'inférence sur les extrémités temporelles d'un processus ; ceci entraîne le retrécissement de la fenêtre temporelle allouée pour son exécution. Les résultats d'une telle démarche peuvent être utilisés pour affiner les caractéristiques des ordonnancements réalisables (ensemble des solutions respectant les contraintes du problème), ou détecter des infaisabilités si, compte tenu de toutes les contraintes, aucune solution n'est possible.

Cette approche s'inscrit dans le concept de l'analyse sous contraintes [ERS 91, LOP 92, ERS 93] qui est d'autant plus profitable et efficace que le problème est fortement contraint. Cette propriété est efficace dans toute démarche fondée sur la propagation des contraintes en tant que moyen de simplification d'un problème de planification [BAB 94a, BAB 94b, BAB 95a, BAB 95b]).

L'approche proposée se présente comme un processus de déduction logique, puisqu'elle met en oeuvre des règles d'inférence ; les contraintes peuvent donc facilement être définies comme des relations sur les variables de décision. Ces règles permettent d'affiner les valeurs des dates limites C_i (commencement) et F_i (fin) initialisées, respectivement, à r_i et d_i, et donc caractériser les ordonnancements réalisables.

D'après l'énoncé de notre problème donné au §1, on voit qu'on a affaire à deux types de contraintes :
— des contraintes temporelles portant sur des positions relatives entre intervalles de temps
— et des contraintes induites par des disjonctions entre intervalles d'exécution de certaines tâches et qui cachent donc des limitations de ressources.
Au début, nous n'allons considérer que les contraintes temporelles. Les contraintes de disjonction seront incluses, ultérieurement, une par une dans le graphe qui modélise le problème.
Afin d'étudier la cohérence de nos contraintes, nous associons à notre problème le graphe suivant :
— un sommet correspond à un commencement (C_x) ou à une fin (F_x) d'un processus $x \in X$.
— chaque sommet est étiqueté d'un intervalle représentant la marge de réalisation de l'évènement qu'il représente.
— un sommet C_x est relié à un sommet F_x (du même processus x) par un arc étiqueté par la durée P_x de x.
— un sommet C_x (resp F_x) d'un processus x est relié à un sommet C_y (resp F_y) ou à un sommet F_y (resp C_y) d'un processus y, s'il existe une contrainte entre x et y qui se traduit par une liaison entre leur débuts et/ou leur fins.
Cette liaison entre les débuts et/ou les fins des processus se répercutent sur leur intervalles respectifs, suite à un déclenchement d'une règle d'inférence correspondante à la contrainte qui relie les deux processus. Une actualisation des bornes de ces intervalles est réalisée à chaque propagation de l'une de ces règles d'inférence. Si le système de contraintes est cohérent, alors cette actualisation se termine. Autrement on "boucle".
Nous avons mis au point les règles d'inférence suivantes, dont chacune correspond à l'une des contraintes que nous avons énoncées dans l'introduction. L'algorithme qui active l'ensemble de ces règles, et qui prend donc en considération l'ensemble des données du problème, sera énoncé par la suite.

a) x avant y : $\Longrightarrow$ $F_x \leq C_y$, d'où la structure graphique correspondante :



Initialement $\underline{C_x} := r_x, \quad \overline{C_x} := d_x - P_x, \quad \underline{F_x} := r_x + P_x, \quad \overline{F_x} := d_x$

$\underline{C_y} := r_y, \quad \overline{C_y} := d_y - P_y, \quad \underline{F_y} := r_y + P_y, \quad \overline{F_y} := d_y.$

D'où les règles d'inférence RIAV :

$$RIAV \begin{cases} \max(\overline{C_y}, \overline{F_x}) + P_y > \overline{F_y} & \longrightarrow \text{ incohérence } (\Longrightarrow stop) \\ \max(\overline{C_y}, \overline{F_x}) + P_y \leq \overline{F_y} & \Longrightarrow \begin{array}{l} \text{actualiser les bornes} \\ \text{selon le système SYS-AV} \end{array} \end{cases}$$

$$SYS\text{-}AV \begin{cases} \underline{C_y} := \max(\underline{F_x}, \underline{C_y}), & \overline{C_y} := \max(\overline{F_x}, \overline{C_y}) \\ \underline{F_y} := \underline{C_y} + P_y, & \overline{F_y} := \overline{C_y} + P_y. \end{cases}$$

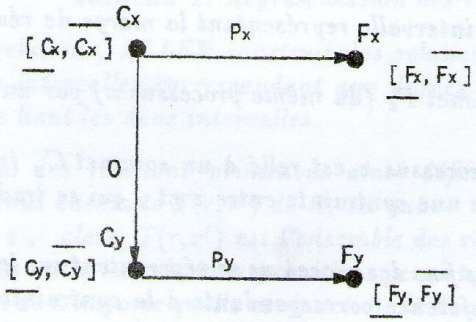
Dans la première règle de RIAV, nous spécifions que la tâche x se termine au plus tard à $\overline{F_x}$. Si y commence au plus tard à $\max(\overline{C_y}, \overline{F_x})$, son exécution dure jusqu'à $\max(\overline{C_y}, \overline{F_x}) + P_y$. Si cette date dépasse la date limite de fin de y, alors il y a une incohérence, et x ne peut être avant y.

Dans le cas contraire, nous réactualisons les valeurs des bornes des intervalles selon le système SYS-AV. Ceci entraîne une compression de ces intervalles.

Comme une compression en déclenche une autre, le processus s'arrête quand il n'y a plus de modification des valeurs C_x et F_x, $\forall x \in X$; c'est-à-dire, entre deux étapes $i-1$ et i de réactualisation de ces valeurs, nous devons avoir : $C_x^i = C_x^{i-1}$ et $F_x^i = F_x^{i-1}$, $\forall x \in X$.

Une actualisation "infinie" de ces valeurs correspond à l'existence d'un circuit ; donc, à l'incohérence des contraintes.

b) x commence-comme y : $\Longrightarrow$ $C_x = C_y$, auquel correspond la structure graphique suivante :



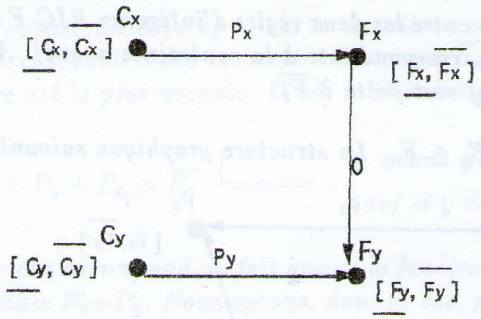
et la règle d'inférence RIC-C :

$$RIC\text{-}C \begin{cases} \max(\underline{C_x}, \underline{C_y}) + P_x \leq \overline{F_x} \wedge \max(\underline{C_x}, \underline{C_y}) + P_y \leq \overline{F_y} \\ \min(\overline{C_x}, \overline{C_y}) + P_x \leq \overline{F_x} \wedge \min(\overline{C_x}, \overline{C_y}) + P_y \leq \overline{F_y} \end{cases} \Longrightarrow \begin{array}{l} \text{actualiser les bornes} \\ \text{selon le système} \\ SYS\text{-}CC \end{array}$$

$$SYS\text{-}CC \begin{cases} \underline{C_x} := \max(\underline{C_x}, \underline{C_y}), & \underline{C_y} := \underline{C_x} \\ \overline{C_x} := \min(\overline{C_x}, \overline{C_y}), & \overline{C_y} := \overline{C_x} \\ \underline{F_x} := \underline{C_x} + P_x, & \overline{F_x} := \overline{C_x} + P_x \\ \underline{F_y} := \underline{C_y} + P_y, & \overline{F_y} := \overline{C_y} + P_y. \end{cases}$$

Cette règle indique que si les processus x et y commencent au plus tôt à $\max(\underline{C_x}, \underline{C_y})$ et au plus tard à $\min(\overline{C_x}, \overline{C_y})$ et ne dépassent pas leur date limite respective, alors la contrainte commence-comme est réalisable et nous pouvons actualiser les valeurs de C et de F (pour x et pour y) tel qu'il est indiqué dans le système SYS-CC.

c) x *finit-comme* y : $\Longrightarrow F_x = F_y$



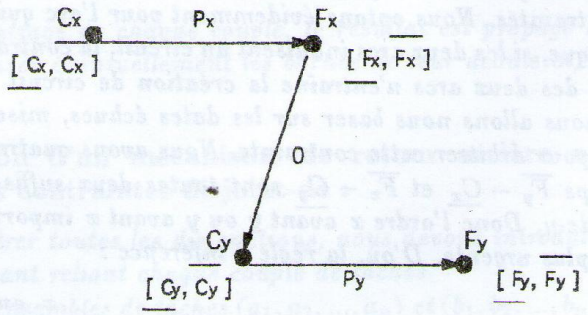
D'où les règles d'inférence RIF-C :

$$\begin{cases} \min(\overline{F_x}, \overline{F_y}) < \max(\underline{F_y}, \underline{F_x}) \models\!= \textit{ incohérence} \\ \\ \textit{RIF-C} \begin{cases} \\ \min(\overline{F_x}, \overline{F_y}) - P_x \geq \underline{C_x} \ \wedge \ \min(\overline{F_x}, \overline{F_y}) - P_y \geq \underline{C_y} \end{cases} \models\!= \begin{array}{l} \textit{actualiser les bornes} \\ \textit{selon le système} \\ \textit{SYS-FC.} \end{array} \end{cases}$$

$$\textit{SYS-FC} \begin{cases} \overline{F_x} := \min(\overline{F_x}, \overline{F_y}), & \overline{F_y} := \overline{F_x} \\ \underline{F_x} := \max(\underline{F_x}, \underline{F_y}), & \underline{F_y} := \underline{F_x} \\ \overline{C_x} := \overline{F_x} - P_x, & \underline{C_x} := \underline{F_x} - P_x \\ \overline{C_y} := \overline{F_y} - P_y, & \underline{C_y} := \underline{F_y} - P_y. \end{cases}$$

La première règle de RIF-C spécifie que si la date de fin au plus tard de l'une des tâches est plus petite que la date de fin au tôt de l'autre tâche, alors il y a une incohérence (qui indique que la contrainte finit-comme ne peut se réaliser).
La deuxième règle de RIF-C indique que si x et y ont une fin commune, ce qui entraîne deux débuts respectifs (dépendant de la durée de chaque tâche), alors, si ces deux débuts sont respectivement après ou égaux aux dates de début au plus tôt de chaque tâche, le processus est consistant et nous pouvons actualiser les bornes des intervalles tel qu'il est mentionné dans le système SYS-FC.

d) y *commence-quand-finit* x : $\Longrightarrow F_x = C_y$. *D'où la structure graphique :*



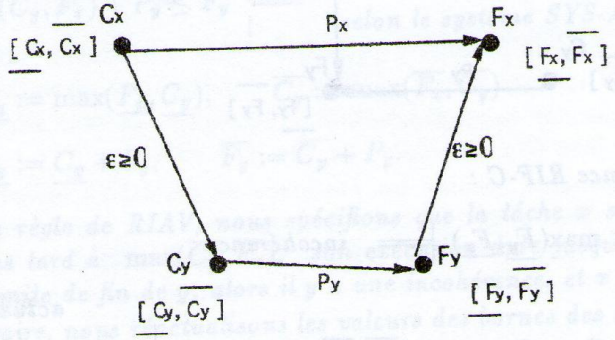
La règle d'inférence RIC-F correspondante est :

$$\textit{RIC-F} \begin{cases} \\ \underline{C_y} \leq \overline{F_x} \ \wedge \ \overline{F_x} + P_y \leq \overline{F_y} \end{cases} \models\!= \begin{array}{l} \textit{actualiser les bornes} \\ \textit{selon le système SYS-CF.} \\ \textit{SYS-CF} \end{array}$$

$$\textit{SYS-CF} \begin{cases} \underline{C_y} := \underline{F_x}, & \overline{C_y} := \min(\overline{C_y}, \overline{F_x}) \\ \underline{F_y} := \underline{C_y} + P_y, & \overline{F_y} := \overline{C_y} + P_y. \end{cases}$$

La partie condition de RIC-F ($\underline{C_y} \leq \overline{F_x}$) spécifie qu'à la fin au plus tard de x, y doit être déjà prête, et que si elle débute au plus tard à cet instant, son exécution ne provoque pas de retard ($\overline{F_x} + P_y \leq \overline{F_y}$). Dans ce cas, on réactualise les bornes des intervalles selon les formules du système SYS-CF.
Remarquons la différence qui existe entre les deux règles d'inférence RIC-F (correspondante à la contrainte commence-quand-finit) et RIAV (correspondante à la contrainte avant). Au cours de la formalisation de RIAV nous n'avons pas exigé que y soit prête à $\overline{F_x}$.

e) $\underline{x \text{ inclut } y}$: $\Longrightarrow$ $C_x \leq C_y < F_y \leq F_x$. La structure graphique suivante schématise cette liaison :



D'où la règle d'inférence :

$$RICLU \quad P_y \leq \overline{F_x} - \underline{C_x} \;\Big|\!\!=\!\!=\!\!=\; \begin{array}{l}\text{actualiser les bornes}\\ \text{selon le système } SYS\text{-}CLU\end{array}$$

$$SYS\text{-}CLU \begin{cases} \underline{C_y} := \max(\underline{C_y}, \underline{C_x}), & \underline{F_y} := \underline{C_y} + P_y \\ \overline{F_y} := \min(\overline{F_x}, \overline{F_y}), & \overline{C_y} := \overline{F_y} - P_y. \end{cases}$$

La condition $P_y \leq \overline{F_x} - \underline{C_x}$ exprime que la durée de y ne doit dépasser l'amplitude de la fenêtre de temps allouée à x. Dans le cas contraire, une incohérence est détectée ; elle correspond au fait que la phase d'exécution de y ne peut être pendant la phase d'exécution de x.
Si la condition ainsi imposée est vérifiée, on réactualise les bornes des intervalles comme il est indiqué dans le système SYS-CLU.

f) $\underline{x \text{ disjoint-de } y}$: $\Longrightarrow$ x avant y $\vee$ y avant x.

Indépendamment des dates de disponibilité et échues des tâches x et y, arbitrer cette contraintes revient à introduire l'arc $\overrightarrow{F_x C_y}$ ou l'arc $\overrightarrow{F_y C_x}$ dans le graphe qui s'est constitué au fur et à mesure de l'arbitrage des autres contraintes. Nous optons évidemment pour l'arc qui n'induit pas de circuit dans le graphe global. Il va de soi que, si les deux arcs induisent un circuit, la contrainte disjoint-de est irréalisable.

Supposons qu'aucun des deux arcs n'entraîne la création de circuit dans le graphe représentant le problème. Dans ce cas, nous allons nous baser sur les dates échues, mises à jour durant l'arbitrage des contraintes précédentes, pour arbitrer cette contrainte. Nous avons quatres cas :

(1) Les fenêtres de temps $\overline{F_y} - \underline{C_x}$ et $\overline{F_x} - \underline{C_y}$ sont toutes deux suffisantes pour contenir les phases d'exécution de x et de y. Donc l'ordre x avant y ou y avant x importe peu. Mais, nous avantageons toujours la tâche la plus urgente. D'où, la règle d'inférence :

$$DISJ\text{-}1 \left\{ \overline{F_y} - \underline{C_x} \geq P_x + P_y \;\wedge\; \overline{F_x} - \underline{C_y} \geq P_x + P_y \;\Big|\!\!=\!\!=\!\!=\; \begin{array}{l} x \text{ avant } y \\ \text{si } x \text{ est plus urgente} \\ \text{que } y \text{ et inversement.} \end{array} \right.$$

(2) Le cas suivant est le premier cas complémentaire du cas précédent : la fenêtre de temps $\overline{F_y} - \underline{C_x}$ est insuffisante pour contenir l'exécution de x puis celle de y (dans cet ordre). Dans ce cas, il faut exécuter y avant x, à moins que la date de fin de x au plus tard soit la plus petite. D'où la deuxième règle d'inférence de disjonction :

$$DISJ\text{-}2 \quad C_x + P_x + P_y > \overline{F_y} \;\Big|\!\!=\!\!=\; \begin{array}{c} y \text{ avant } x \\ \text{sauf si } x \text{ est plus urgente.} \end{array}$$

(3) Ce cas est l'opposé du cas (2) et est le deuxième cas complémentaire du cas (1) : l'exécution de y avant x est irréalisable dans la fenêtre de temps $\overline{F_x} - C_y$. Donc, il faut exécuter x avant y, sauf si cette dernière est la plus urgente. D'où, la troisième règle d'inférence de disjonction :

$$DISJ\text{-}3 \quad C_y + P_y + P_x > \overline{F_x} \;\Big|\!\!=\!\!=\; \begin{array}{c} x \text{ avant } y \\ \text{sauf si } y \text{ est plus urgente.} \end{array}$$

(4) Le quatrième cas correspond au fait que ni la fenêtre $\overline{F_x} - C_y$, ni la fenêtre $\overline{F_y} - C_x$ est suffisante pour contenir la phase $P_x + P_y$. Nous optons, dans ce cas, pour l'ordre qui entraine le moins de détérioration de l'objectif fixé ($\min T_{\max}$) dans l'espace des solutions accessibles par notre heuristique : x avant y, si $C_x + P_x + P_y - \overline{F_y} < C_y + P_y + P_x - \overline{F_x}$. Si ces deux quantités sont égales, nous exécutons en premier la tâche de plus petite date de fin au plus tard. En cas d'une deuxième égalité, nous exécutons la tâche la moins longue en premier. D'où les règles d'inférence suivante :

$$DISJ\text{-}4 \left\{ C_x + P_x + P_y > \overline{F_y} \wedge C_y + P_y + P_x > \overline{F_x} \right. \;\Big|\!\!=\!\!=\; \begin{array}{c} x \text{ avant } y \\ \text{si } C_x - \overline{F_y} < C_y - \overline{F_x} \\ \text{et inversement.} \end{array}$$

$$DISJ\text{-}5 \left\{ \begin{array}{l} C_x + P_x + P_y > \overline{F_y} \wedge C_y + P_y + P_x > \overline{F_x} \\ \wedge \; C_x - \overline{F_y} = C_y - \overline{F_x} \end{array} \right. \;\Big|\!\!=\!\!=\; \begin{array}{c} x \text{ avant } y \\ \text{si } x \text{ est plus urgente} \\ \text{et inversement.} \end{array}$$

$$DISJ\text{-}6 \left\{ \begin{array}{l} C_x + P_x + P_y > \overline{F_y} \wedge C_y + P_y + P_x > \overline{F_x} \\ \wedge \; C_x - \overline{F_y} = C_y - \overline{F_x} \wedge \overline{F_x} = \overline{F_y} \end{array} \right. \;\Big|\!\!=\!\!=\; \begin{array}{c} x \text{ avant } y \\ \text{si } x \text{ est plus urgente} \\ \text{et inversement.} \end{array}$$

Dans tout problème d'ordonnancement, quand la contrainte de disjonction est imposée, elle porte généralement sur plusieurs couples de tâches. Le problème qui se pose alors consiste à définir une politique de choix d'un couple précis que nous devons arbitrer prochainement.

Dans notre approche, nous définissons la règle suivante :

— *après l'arbitrage d'un couple donné, classer toutes les tâches concernées par la contrainte disjoint-de dans l'ordre croissant de leur dates de fin au plus tard,*

— *choisir le couple dont l'une des tâches a la date de fin la plus petite.*

Après l'arbitrage de chaque couple, le résultat est propagé dans l'ensemble des tâches du problème, afin de réactualiser éventuellement les bornes de leur débuts et/ou de leur fins.

2.3.2 Définition d'un mécanisme de retour arrière nécessaire en cas de blocage dans l'arbitrage des contraintes disjoint-de :

Afin d'arbitrer toutes les disjonctions, nous devons introduire successivement les arcs correspondant aux relations avant reliant chaque couple de tâches.

Soient les deux ensembles de tâches $(a_1, a_2, ..., a_n)$ et $(b_1, b_2, ..., b_n)$ tels que a_1 disjoint-de b_1, a_2 disjoint-de b_2, ... etc.

Admettons que nous ayons arbitré les $(i-1)$ premiers couples, et supposons qu'en introduisant aussi bien l'arc $\overrightarrow{F_{a_i} C_{b_i}}$ que l'arc $\overrightarrow{F_{b_i} C_{a_i}}$ (pour arbitrer le ième couple), nous créons un circuit.

Dans ce cas, nous revenons sur l'arbitrage du couple (a_{i-1}, b_{i-1}) même si ceci entraine un retard, pourvu que le nouvel arbitrage n'induise pas de circuit. Si par contre la remise en question de l'arbitrage du couple (a_{i-1}, b_{i-1}) entraine la création d'un circuit, nous reconsidérons l'arbitrage du couple (a_{i-2}, b_{i-2}) ; et ainsi de suite.

Si aucun arbitrage des couples précédant le couple (a_i, b_i) ne peut être remis en question, alors l'arbitrage de (a_i, b_i) est impossible. Ce qui prouve l'incohérence des contraintes entre elles.

Le problème qui peut se poser dans ce processus de retour arrière est qu'à un moment donné la masse de données manipulées risque de devenir assez grande, au point où sa gestion devient très fastidieuse. Pour éviter ce problème, nous envisageons de mémoriser à chaque étape l'ensemble des arcs susceptibles de créer des circuits dans le graphe et d'en former une sorte d'ensemble tabou que nous ne devons essayer qu'en dernier recours. Ceci permet, entre autres, de définir un mécanisme de filtrage, et donc d'accélérer le backtracking.

Ce que nous venons d'exposer depuis le début du §2.3 est la manière d'agir face à chaque contrainte donnée dans l'énoncé du problème formulé au §1. Nous avons énoncé chaque règle d'inférence séparément pour schématiser le processus de propagation correspondant à chaque contrainte. Afin de tester la cohérence globale des données du problème posé, il est nécessaire de prendre en considération l'ensemble des tâches et l'ensemble des contraintes qui les relient (y compris leur dates de disponibilité et échues). Dans ce but, nous avons mis au point un module qui, pour chaque couple de tâches, assure le déclenchement de la règle d'inférence correspondante à la contrainte qui les relie et la propagation du résultat de cette inférence. Le traitement détaillé est donné dans la procédure Actualisation suivante :

Procédure Actualisation ;
Début
Pour tout processus $x \in X$ *faire*
$\underline{C_x}^{\circ} := r_x$; $\overline{C_x}^{\circ} := d_x - P_x$; $\underline{F_x}^{\circ} := r_x + P_x$; $\overline{F_x}^{\circ} := d_x$;
fait ;
$i := 1$; $\neg stop$;
Tant que $\neg stop$ *faire*
Considérer les couple de tâches dans leur ordre lexicographique ;
Pour tout couple $(x, y) \in X^2$ *faire*
en fonction de la contrainte qui relie x *à* y, *activer la règle d'inférence correspondante ;*
à l'étape i *actualiser les valeurs de* $\underline{C}^i, \overline{C}^i, \underline{F}^i, \overline{F}^i$ *(de* x *et de* y*) en fonction de leurs valeurs à l'étape* $(i-1)$;
fait ;
Si $\underline{C_x}^i = \underline{C_x}^{i-1}$ *(resp* $\overline{C_x}^i = \overline{C_x}^{i-1}, \underline{F_x}^i = \underline{F_x}^{i-1}, \overline{F_x}^i = \overline{F_x}^{i-1}$*)* $\forall x \in X$
alors stop ;
fsi ;
$i := i + 1$;
Si i *très grand alors stop ;*
{*on revient sans cesse à l'actualisation de certaines valeurs ; donc, il existe un circuit (contraintes non cohérentes)*} ;
fsi ;
fait ;
Arbitrer les contraintes disjoint-de ;
Fin.

2.3.3 Nécessité de la mise au point d'un module de relaxation :

Les règles d'inférence qui ont été décrites depuis le début du §2.3, ainsi que la procédure Actualisation, expriment toutes un moyen d'actualisation des caractéristiques de réalisabilité des solutions du problème posé. Nous entendons par solution réalisable un ensemble d'encadrements du début et de la fin de chaque tâche qui vérifient l'ensemble des contraintes du problème.
Comme nous l'avons déjà montré, la stratégie générale de l'analyse sous contrainte a pour objectif de parvenir à un contexte où aucune caractéristique nouvelle de la (ou des) solution(s) du problème ne peut être obtenue. Pour un problème posé, il n'y a aucune garantie de la consistance de l'ensemble de ses

données ; en particulier, quand celui-ci est "trop contraint". Dans ce cas, tôt ou tard une incohérence peut apparaître au cours de la résolution. Quand cette incohérence est détectée suite à une actualisation des dates limites, une absence de toute politique de relaxation de certaines données ou contraintes peut entraîner un blocage facile à éviter au cas où un module de relaxation a été mis au point.

Remarque :

Le concept décrit par la procédure Actualisation est polynomial, puisqu'il est basé sur la propagation de contraintes dans un réseau d'intervalles [DAV 87]. Il revient, en fait, à déterminer un chemin dans un graphe, ce qui est habituellement réalisé en $O(n^2)$. Le processus d'inférence associé à cette propagation est sain et complet. La structure graphique adoptée remplace la notion de graphe potentiel-tâche couramment utilisé en ordonnancement.

En cas d'exécution de la procédure Actualisation (ce que nous avons constaté au cours de son implémentation), nous pouvons nous en apercevoir que l'un des cas d'incohérence du système est exprimé par les deux propositions triviales suivantes qui enrichissent notre ensemble de règles d'inférence :

Proposition 1 :

S'il existe un triplet de processus (a, b, c) tel que :

$$\begin{cases} a \text{ commence-comme } b \\ c \text{ finit-comme } b \\ a \text{ avant } c \\ P_a + P_c > P_b \end{cases}$$

alors, les contraintes imposées dans le problème sont antagonistes.

Remarque :

Si le couple (a, c) n'est pas relié par la contrainte avant, les contraintes commence-comme et finit-comme n'induisent pas d'incohérence même si $P_a + P_c > P_b$.

Proposition 2 :

S'il existe un triplet de processus (a, b, c) tel que :

$$\begin{cases} b \text{ commence-quand-finit } a \\ b \text{ avant } c \quad (\text{ou } c \text{ commence-quand-finit } b) \\ c \text{ avant } a. \end{cases}$$

alors, les contraintes du problème sont antagonistes.

L'existence d'un tel triplet induit un circuit dans le graphe global représentatif du problème.

Proposition 3 :

a inclut b est réalisable si :

$\forall c, d \in X$: *si* $(c \text{ avant } a \ \wedge \ a \text{ avant } d)$ *alors* $(c \text{ avant } b \ \wedge \ b \text{ avant } d)$.

Démonstration :

Si a inclut b, alors il existe sûrement un arc $\overrightarrow{C_a C_b}$ qui relie début(a) à début(b) et un arc $\overrightarrow{F_b F_a}$ qui relie fin(b) à fin(a), tel que nous l'avons déjà explicité dans les représentations graphiques précédentes. D'autre part, si c avant a, alors fin(c) est relié à début(a) par l'arc $\overrightarrow{F_c C_a}$; donc, il est obligatoirement relié à début(b). Ceci prouve que b ne peut commencer qu'après la fin de c.
Le même raisonnement peut être adopté pour démontrer que b est avant d. □

2.3.4 Structuration de l'implémentation de l'approche proposée :

Cette implémentation comporte :

(1) un module comprenant :
- des faits invariants au cours du temps concernant les caractéristiques de chaque processus (son début, sa durée, sa fin et ses relations avec les autres processus),
- des faits susceptibles d'évoluer au cours du raisonnement et qui caractérisent les solutions réalisables. Nous désignons essentiellement :
  - les bornes des intervalles associés au début et à la fin de chaque processus,
  - certaines contraintes entre certains processus (particulièrement les relations de précédence qui apparaissent suite à un arbitrage des contraintes de disjonction).

(2) un bloc de contrôle qui guide la sélection d'une règle d'inférence en fonction de la contrainte propagée,

(3) un module caractérisant le processus de propagation des valeurs extrêmes des bornes des intervalles mises à jour après l'activation d'une règle d'inférence,

(4) un bloc de déduction, basé sur le principe du backtracking, pour générer de nouvelles relations entre certaines tâches et caractériser ainsi l'arbitrage des contraintes de disjonction,

(5) un module de relaxation qui permet d'éviter certaines situations de blocage.

La réalisation de cette implémentation donne un ensemble de solutions réalisables, ou signale une incohérence au sein des données du problème posé.

3. Etablissement d'un ordonnancement :

Dans ce type de problème d'ordonnancement, c'est-à-dire quand les dates de disponibilité et échues sont données, le critère mis en jeu pour établir un ordonnancement se base sur la date de fin d'exécution de chaque tâche. Cette date de fin peut être inférieure ou égale à la date échue ($F_x \leq d_x$) ce qui correspond à un service meilleur ou semblable à celui auquel on s'attendait, ou bien elle peut être après la date échue ($F_x > d_x$), dans ce cas on parle de retard d'exécution.
On définit le retard d'une tâche x par :

$$T_x = \max(0, F_x - d_x) \qquad \forall x \in X.$$

Les études réalisées dans ce domaine visent le plus souvent la minimisation de l'un des critères suivants :

le retard moyen : $\overline{T} = \frac{1}{n} \sum_x T_x, \quad où \quad n = |X|,$

le retard maximal : $T_{\max} = \max_x T_x \qquad x \in X,$

le nombre de tâches en retard : $N_T = \sum_x \delta(T_x)$

$$où \quad \delta(i) = \begin{cases} 1 & si\ i > 0 \\ 0 & autrement. \end{cases}$$

De notre part, nous cherchons à déterminer un ordonnancement qui minimise le retard maximal des tâches ; c'est-à-dire, nous cherchons début(x) que nous notons t_x, qui donne au critère $T_{\max}$ sa meilleure valeur dans l'espace des solutions accessibles par notre heuristique.

Afin d'atteindre cet objectif, nous allons nous baser sur la propriété de dominance de certaines solutions. Cette relation de dominance permet de comparer des séquences vis-à-vis de leur réalisabilité. Il est judicieux dans ce cas de se baser sur les dates limites d'achèvement des tâches. Une solution dominante ainsi déterminée minimise certainement le critère fixé ($\min T_{\max}$) [BAK 74].
Dans ce type de problème, la détermination d'une séquence dominante de tâches se base sur l'arbitrage suivant :
Soient x et y deux tâches telles que $r_x < r_y$ et $d_x < d_y$. L'intervalle $d_y - r_x$ alloué à la séquence x avant y est plus important que l'intervalle $d_x - r_y$ alloué à la séquence y avant x. D'où, on voit que la séquence x avant y domine la séquence y avant x.
Ce raisonnement est valable et nécessaire quand on a affaire à un problème d'ordonnancement sur une seule ressource. Le problème du nombre de ressources ne se pose pas dans notre étude.

En généralisant cette politique, nous allons déterminer des séquences de tâches qui respectent l'ensemble des contraintes imposées et donnent au critère fixé sa meilleur valeur dans le domaine des solutions réalisables..

Notre démarche se résume comme suit :

- *A l'état initial, nous choisissons les tâches x de dates de disponibilité minimales (r_x minimale) et dont les ensembles des prédécesseurs sont vides. Parmi toutes les tâches qui vérifient ces conditions, nous considérons la plus urgente (de d_x minimale).*
- *Nous prévoyons une liste d'évènements dans notre système de résolution. La fin d'exécution de la tâche x choisie dans l'étape précédente constitue un évènement qu'il faut mémoriser dans la liste des évènements.*
- *S'il existe des tâches y reliées à la tâche choisie x par la contrainte y commence-comme x, elles sont prises dans l'ordre de leur dates échues. Le début de chaque tâche y, ainsi reliée à x, est alors fixé égal à début(x).*
  La fin d'exécution de chaque tâche y (où y commence-comme x) constitue un évènement qu'il faut également mémoriser dans la liste des évènements.
- *S'il existe des tâches z reliées à la tâche choisie x par la contrainte z finit-comme x, alors pour respecter cette contrainte, le début de chaque tâche z est fixé égal à $fin(x) - P_z$ qui constitue en lui même un évènement que nous devons mémoriser dans la liste des évènements.*
- *Après tout choix d'une tâche à exécuter, la liste des évènements est ordonnée, et nous considérons l'évènement de plus petite date.*
- *Dans une prochaine étape de cet algorithme, l'instant actuel t est égal à la plus petite date d'un évènement choisi dans la liste des évènements. A cet instant t, nous choisissons parmi les tâches non encore ordonnées celle qui vérifie les conditions fixées au premier point ; c'est-à-dire, celle qui est déjà prête ($r_x \leq t$), dont tous les prédécesseurs ont été ordonnés ($\Gamma^-(x) = \phi$) et dont la date échue est la plus petite.*

L'ensemble de ces opérations est détaillé dans la procédure Ordonnancement suivante dont l'objectif principal est de déterminer début(x) noté t_x $\forall x \in X$, tout en respectant les contraintes binaires données au §1.

Préalablement, notons par :

OR : l'ensemble des tâches ordonnées,

$\overline{OR}$: l'ensemble des tâches non encore ordonnées,

LIST : la liste des évènements. Un évènement correspond à un début ou à une fin d'une tâche,

PLAC : l'ensemble des tâches qu'on réussit à ordonner en même temps (dans la boucle principale (externe) tant que de la procédure Ordonnancement suivante),

et t : l'instant actuel.

Initialement, nous avons : $OR := \phi$, $\overline{OR} := X$, $LIST := \phi$, $PLAC := \phi$, et $t := \min_{x \in X} r_x$;

Avant d'énoncer la procédure Ordonnancement, remarquons que la procédure Actualisation aurait dû être terminée par les instructions suivantes :

Pour tout $x \in X$ faire
$r_x := C_x$; *$d_x := \overline{F_x}$;*
fait ;

Procédure Ordonnancement ;
Début
Tant que $OR \neq X$ faire

à l'instant t, choisir $x \in \overline{OR}$ *telle que :*

$$(*)\begin{cases} r_x \leq t\ ; \\ \Gamma^-(x) = \phi\ ; \\ d_x = \min\limits_{\substack{i \in \overline{OR} \\ \Gamma^-(i)=\phi}} d_i \end{cases}$$

Si un tel x n'existe pas alors
choisir le prochain évènement dans LIST ;
Si un tel évènement n'existe pas alors
choisir le prochain plus petit r_x, $x \in \overline{OR}$;
fsi ; *Choisir* $x \in \overline{OR}$ *qui vérifie les conditions* $(*)$;
fsi ;
$t_x := t$;
joindre l'évènement $fin(x) = t_x + P_x$ *à LIST ;*
Tant que $\exists y \in \overline{OR}$ / *y commence-comme x faire*
$t_y := t_x$;
joindre $fin(y)$ *à LIST ;*
Tant que $\exists z \in \overline{OR}$ / *z finit-comme y* *z finit-comme x*
faire
$t_z := fin(y) - P_z$; *(resp* $t_z := fin(x) - P_z$;*)*
joindre t_z *à LIST ;*
fait ;
remettre à jour OR, $\overline{OR}$*, LIST et PLAC ;*
$t :=$*le prochain plus petit évènement dans LIST ;*
fait ;
fait ;
Fin.

Après chaque affectation d'une tâche x, la remise à jour de OR, $\overline{OR}$ *et PLAC se fait comme suit :*

Procédure Remise-à-jour ;
Début
$\overline{OR} := \overline{OR} - \{x\}$;
$OR := OR \cup \{x\}$;
$PLAC := PLAC \cup \{x\}$;
Tant que $PLAC \neq \phi$ *faire*
prendre $y \in PLAC$;
Pour tout $z \in \overline{OR}$ / *y avant z faire*
$r_z := \max\{r_z, fin(y)\}$;
$\Gamma^-(z) := \Gamma^-(z) - \{y\}$;
fait ;
$PLAC := PLAC - \{y\}$;
fait ;
Fin.

Le choix d'un évènement de LIST se fait en suivant les instructions suivantes :

Début
Si $LIST \neq \phi$ *alors*
$t := \min\{t_e$ / $t_e \in LIST\}$;
$\{t_e$ *est l'instant de réalisation d'un évènement e qui peut être un début ou une fin dans LIST*$\}$;
$LIST := LIST - \{t_e\}$;
fsi ;
Fin.

La remise à jour de LIST par l'adjonction d'un évènement t_{nouv}, où $t_{nouv} = début(x)$ ou $t_{nouv} = fin(x)$, se fait par les instructions suivantes :

Début
  Si $t_{nouv} \notin LIST$ *alors*
    $LIST := LIST \cup \{t_{nouv}\}$;
  fsi ;
Fin.

Le test qui consiste à vérifier si $t_{nouv} \notin LIST$ garantie la minimalité de LIST.

4. Un aperçu sur les résultats de l'implémentation :

Les procédures Actualisation et Ordonnancement ont été implémentées sur une station SUN C 3-260. Dans la procédure Actualisation, le module le plus difficile à réaliser est celui qui arbitre les contraintes de disjonction, et particulièrement la gestion des retours arrière.

Les temps de calcul des deux procédures dépendent étroitement du nombre de tâches et de la structure du graphe adopté : autant celui-ci est dense et le nombre de tâches est grand, autant les temps de calcul augmentent tout en restant largement admissibles. Le tableau 2 suivant donne un aperçu sur ces résultats. Les septs premières colonnes de ce tableau présentent la structure du graphe adopté et les cinqs dernières expriment le comportement des deux procédures respectivement.

Nous remarquons que le nombre maximal d'itérations mis par la procédure Actualisation pour déterminer un ensemble de solutions réalisables ne dépasse pas 10. Néanmoins, nous nous sommes fixés comme nombre maximal d'itérations pour déceler un circuit à 50; ce qui a été réalisé en un temps assez raisonnable (moins de 2 secondes pour des problèmes de tailles moyennes).

Le temps de calcul de la procédure Ordonnancement est beaucoup plus réduit par rapport à celui de la procédure Actualisation. Ceci est totalement normal, puisque cette dernière parcourt le graphe plusieurs fois avant d'atteindre un état stable (caractérisé par la détermination d'un ensemble de solutions réalisables) qui sert à la première pour établir un ordonnancement en une seule étape.

Ce que nous avons retenu en particulier au cours de la réalisation de cette implémentation, est que quand les contraintes binaires ne sont pas antagonistes entre elles, le recours au module de relaxation de contraintes imposées sur les fenêtres de temps est inévitable. Cette relaxation devient de plus en plus nécessaire que la taille du problème augmente, et la structure du graphe représentatif se complique.

Pour certains problèmes réels, une telle relaxation est inconcevable, ou du moins sa réalisation entraîne un coût supplémentaire dans la réalisation du projet global. D'autre part, certaines tâches peuvent être retardées plus que d'autres. Signalons aussi, qu'un type de liaisons entre un ensemble de tâches peut être souhaitable sur le plan technique, mais irréalisable sur le plan structurel global du problème.

En prenant en considération tous ces facteurs, il est bien visible que le module de relaxation ne peut être exécuté qu'en interactif. Cette aptitude permet à l'utilisateur d'analyser les conséquences de telles relaxations et d'émettre son avis sur leur admissibilité ou leur rejet.

5. Conclusion :

Dans cette étude, nous avons traité un problème d'ordonnancement où les tâches sont soumises à un ensemble de contraintes, entre autres, celles qui portent sur les fenêtres de temps de leur exécution.

Nous avons agi en deux étapes. Dans la première, notre objectif été de tester la cohérence de l'ensemble des contraintes du problème. L'approche que nous avons adoptée nous a permis de déterminer un ensemble de solutions réalisables.

A base de cet ensemble, nous avons essayé, en seconde étape, de dater les processus. L'ordonnancement ainsi établi donne la meilleure valeur à notre critère dans l'espace des solutions accessibles par notre heuristique.

La stratégie adoptée dans la première étape a été présentée sous forme d'un processus d'inférence qui affine progressivement les caractéristiques des solutions réalisables du problème posé. Elle fait intéragir les différentes contraintes; ce qui, d'une part, entraîne le resserrement des intervalles des dates limites, et d'autre part, permet, compte tenu des liaisons déjà existantes entre les processus, de définir d'autres

relations de succession entre certains de ces processus afin d'arbitrer un type spécifique de contraintes (de disjonction).
Nous avons vu qu'il été nécessaire de concevoir un module de relaxation destiné à modifier certaines données du problème lorsque, compte tenu des données initiales et de l'ensemble des contraintes, une incohérence est détectée.
Ceci permet d'envisager une approche hiérarchisée pour satisfaire les contraintes globales du problème.

En deuxième étape et à base de l'ensemble des solutions réalisables, nous avons défini un critère de dominance entre ces solutions. Ce critère de dominance consistait à privilégier à chaque étape la tâche de date échue minimale.
L'ordonnancement ainsi déterminé respecte la plupart des contraintes et donne au critère établi sa meilleure valeur dans l'espace des solutions accessibles par notre heuristique.

Ce que nous pouvons retenir de la méthode adoptée en première étape, est qu'elle peut être exploitée dans un contexte statique où elle permet de déterminer un ensemble de solutions réalisables servant de base pour établir un plan de gestion spécifique (donc selon certaines caractéristiques supplémentaires). Elle peut être également utilisée dans un contexte dynamique pour aider à établir un ordonnancement en temps réel, où la mise à jour des bornes des intervalles et l'ajout ou le retrait de certaines relations entre processus se fait selon des faits nouveaux qui arrivent au module de résolution instantanément.

Finalement, nous pouvons affirmer que l'approche adoptée est fort bien extensible pour n'importe quel système de contraintes. Ceci nous encourage à envisager l'enrichissement du système des contraintes binaires considéré, par des contraintes schématisant au mieux les problèmes réels.

Bibliographie

[ALL 81] *J.F. ALLEN : "An interval based representation of temporel knowlege", In Procedings of the 7th IJCAI - Vancouver, Canada, 1981, P221-226.*

[ALL 84] *J.F. ALLEN : "Towards a general theory of action and time", Artificial Intelligence, vol 23, 1984, P123-154.*

[BAB 94a] *M. BABES : "Techniques de propagations de contraintes et de moindre engagement pour résoudre le problème d'emploi du temps". Congré international D'Avignon'94, Palais des congrés, Paris, 30 mai-3juin 1994, P251-263.*

[BAB 94b] *M. BABES : "Concilliation de la R.O à l'I.A pour résoudre un problème d'emploi du temps". Colloque international de l'Université de Damas, du 06 au 12 Nov 94, Damas, Syrie.*

[BAB 95a] *M. BABES, A. QUILLIOT : "Une approche basée sur un concept logique et un formalisme mathématique, afin de résoudre le problème d'emploi du temps". Automatique Productique Informatique Industrielle (APII), vol 29, no 1, 1995, P7-38.*

[BAB 95b] *M. BABES : "Conception d'un logiciel en traitement interactif de problèmes d'emploi du temps et d'ordonnancement". Doctorat d'état, Juin 1995, Université d'Annaba, Algérie.*

[BAK 74] *K.R. BAKER : " Introduction to sequencing and scheduling", John Willey and sons, New-york, 1974.*

[CAR 84] *J. CARLIER : "Problèmes d'ordonnancement à contraintes de ressources : Algorithmes et complexité", Thèse de doctorat d'état, Université de P&M Currie, sept. 1984.*

[DAV 87] *E. DAVIS : "Constraint propagation with interval labels", A.I - 32, 1987, P281-331.*

[ERS 91] *J. ERSCHLER, P. LOPEZ, C. THURIOT : "Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement", Revue d'Intelligence Artificielle, vol 5, no 3, 1991, P7-32.*

[ERS 93] *J. ERSCHLER, G. FONTAN, C. MERCE : "Approche par contraintes en planification et ordonnancement de la production", Automatique Productique Informatique Industrielle (APII), vol 27, no 6, 1993, P669-695.*

[FLO 71] *M. FLORIAN, P. TREPANT, G. McMAHON : "An implicit enumeration algorithm for the machine sequencing problem", Mangnt. Sci., vol 17, no 12, august 1971, P782-792.*

[GHA 89] *M. GHALLAB M : "Représentation et gestion de relations temporelles", Proceeding AFCET R.F.I.A, T1, 1989, P3-22.*

[JAC 55] *J.R JACKSON : "Scheduling a production line to minimize maximum tardiness", Research repport, 43, Mangnt. Sci. Research project, University of California, Los Angeles, 1955.*

[LAG 76] *B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN : "Minimizing maximum lateness on one machine : computational experience and some applications", Statist. Neerlandica, 30, 1976, P25-41.*

[LEN 76] *J.K. LENSTRA : "Sequencing by Enumerative methods", Mathematisch Centrum, Amsterdam, 1976.*

[LOP 92] *P. LOPEZ, J. ERSCHLER, P. ESQUIROL : "Ordonnancement de tâches sous contraintes : Une approche energitique", Automatique Productique Informatique Industrielle (API), vol 26, no 5-6, 1992, P453-481.*

[MUN 70] *R.R. MUNTZ, E.G. COFFMAN : "Preemptive scheduling of real time tasks on multiprocessor systems", ACM, vol 17, no 2, April 1970, P324-338.*

[NIC 91] *P. NICOLAS : "Planification et allocation de ressources avec contraintes temporelles, Un sustème de raisonnement non monotone, Une méthode de génération d'emploi du temps", Thèse de Doctorat d'université, Clermont-Ferrand, Janvier 1991.*

[RIT 88] *J.F. RIT : "Modélisation et propagation de contraintes temporelles pour la planification", Thèse de docteur-ingénieur de l'INPG, 7 Mars 1988.*

[ROY 70] *B. ROY : "Algèbre moderne et théorie des graphes", Dunod, Paris - 1970.*

[SAU 87] *N.W. SAUER, M.G. STONE : "Rational preemptive scheduling", Order, no 4, 1987, P195-206.*

[SAU 89] *N.W. SAUER, M.G. STONE : "Preemptive scheduling of interval orders is polynomial", Order, no 5, 1989, P345-348.*

Structure graphique adoptée							Procédure Actualisation			Procédure Ordonnancement	
Nombre total de tâches	Nombre de couples de tâches reliées par avant	Nombre de couples de tâches reliées par comm-c	Nombre de couples de tâches reliées par finit-c	Nombre de couples de tâches reliées par c-quand-f	Nombre de couples de tâches reliées par inclut	Nombre de couples disjonctifs	Nombre d'étapes (itérations)	Nombre de relaxations de la date échue	Temps de calcul (en S)	Durée du retard maximal	Temps de calcul (en S)
10	4	6	11	2	2	2	3	2	0.08	6	0.03
15	7	12	12	6	2	4	3	7	0.14	10	0.04
20	14	12	16	7	4	4	3	8	0.19	10	0.05
25	20	14	19	9	5	6	4	10	0.35	10	0.06
30	23	17	21	10	5	8	5	12	0.49	17	0.09
35	27	24	23	11	6	11	5	14	0.62	18	0.13
40	28	26	29	13	7	12	5	17	0.82	18	0.23
45	31	26	30	17	7	15	7	21	1.04	22	0.26
50	34	29	33	19	9	17	7	24	1.42	22	0.35

Tableau 2. Représentation des résultats de l'implémentation *des procédures Actualisation et Ordonnancement*