

Une contrainte globale pour l'ordonnançabilité des tâches temps réel dur

Hadrien Cambazard Pierre-Emmanuel Hladik
Anne-Marie Déplanche Narendra Jussien

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
École des Mines de Nantes, LINA FRE CNRS 2729
4 rue Alfred Kastler – BP 20722 – F-44307 Nantes Cedex 3, France
IRCCyN, UMR CNRS 6597
1 rue de la Noë – BP 92101 – F-44321 Nantes Cedex 3, France
H.Cambazard@4c.ucc.ie,
{Pierre-Emmanuel.Hladik,Narendra.Jussien}@emn.fr,
Anne-Marie.Deplanche@irccyn.ec-nantes.fr

Résumé

Résoudre des problèmes de placement de tâches temps réel est un challenge pour les communautés issues de la recherche opérationnelle et de la programmation par contraintes (ppc). La difficulté principale de tels problèmes réside dans la prise en considération des contraintes temporelles spécifiques aux systèmes temps réel. Ces contraintes sont difficilement exprimables pour un solveur et méritent donc un traitement particulier. Dans cet article, nous proposons une approche novatrice qui introduit ces aspects temporels par le biais d'une contrainte globale et donc directement dans les mécanismes de la ppc. Nous finalisons l'étude en montrant à travers de nombreuses expérimentations que cette méthode est viable et efficace.

Abstract

Solving hard real-time task allocation problems is a challenging application for the Operations Research and Constraint Programming communities where the difficulty and novelty lie in timing constraints which occur frequently in real-time systems. In this paper, we address the timing constraints with a global constraint to prune the search space. We experimentally show that the equations expressing the timing constraints are sufficiently easy to solve in practice to embed their resolution within a global constraint.

1 Introduction

Les systèmes temps réel embarqués sont aujourd'hui au cœur de nombreuses applications industrielles. Pour répondre aux nombreuses demandes de traitement, elles sont généralement composées de plusieurs processeurs spécialisés et distribués (interconnectés par un réseau). Par exemple, dans une automobile, de telles applications ont en charge le contrôle de la trajectoire, l'ABS, le contrôle moteur, etc. Les spécifications de ces systèmes sont aussi bien fonctionnelles que non fonctionnelles, ce qui comprend la répartition physique en fonction des ressources et les contraintes temporelles sur les tâches à exécuter. Les contraintes temporelles sont habituellement des échéances sur les dates de terminaison des tâches qui, si elles ne sont pas respectées, peuvent conduire à une défaillance grave de l'application. Dans ce cas, les systèmes sont dits systèmes *temps réel durs* et la prédictibilité temporelle est obligatoire.

Dans cet article, nous traitons du problème qui consiste à assigner des tâches périodiques, préemptives à priorité fixe (une tâche est activée périodiquement et peut être préemptée par une tâche de plus haute priorité) sur un ensemble de processeurs distribués. Une solution correspond à un placement de toutes ces tâches sur les processeurs, de telle manière à ce qu'elles respectent leur échéance.

Le problème de placement de tâches préemptives temps réel durs sur une architecture distribuée est prouvé NP-dur [8] et a déjà été abordé à l'aide de méthodes heuristiques [4], de recuit simulé [15] et d'algorithmes génétiques [4]. Cependant ces techniques sont souvent incomplètes et peuvent ainsi ne pas trouver de solution même s'il en existe. Une nouvelle approche complète est donc utile.

Dans de premiers travaux sur le sujet [2], nous avons proposé une approche basée sur la décomposition de Benders en séparant les contraintes de ressource des contraintes temporelles. Ici, nous proposons d'introduire une contrainte globale qui prend directement en considération les aspects d'ordonnement.

L'article est ainsi organisé de la manière suivante : la partie 2 présente le problème. Un état de l'art et la stratégie de résolution sont introduits dans la partie 3. Pour terminer, des résultats expérimentaux sont affichés dans la partie 4.

2 Description du problème

2.1 L'architecture temps réel

Un système temps réel dur est modélisé par une architecture logicielle (l'ensemble des tâches et des messages) et une architecture matérielle (le support physique d'exécution). Le tableau 1 récapitule l'ensemble des notations que nous introduisons dans la suite.

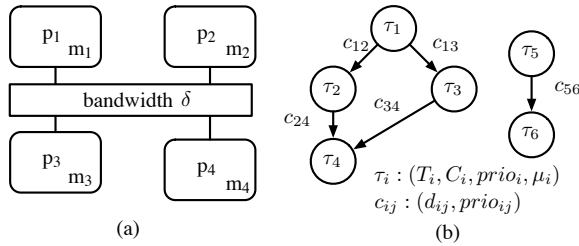


FIG. 1 – Architectures (a) matérielle et (b) logicielle d'un système temps réel

2.1.1 Architecture matérielle

L'architecture matérielle HA est constituée d'un ensemble $\mathcal{P} = \{p_1, \dots, p_k, \dots, p_m\}$ de m processeurs ayant des capacités mémoire μ_k et connectés par un réseau \mathcal{N} . L'architecture est alors donnée par le couple $(\mathcal{P}, \mathcal{N})$. Tous les processeurs de \mathcal{P} ont la même vitesse d'exécution. Le réseau possède une bande passante δ . Dans un travail antérieur [2], nous considérons un réseau à jeton ; ici, nous nous

basons sur un réseau CAN (Control Area Network) en mode désynchronisé, c.-à-d. que l'émetteur d'un message et son récepteur sont considérés comme indépendants. CAN [16] a été développé par l'industrie automobile et est utilisé dans de nombreuses applications temps réel.

2.1.2 Architecture logicielle

Pour modéliser l'architecture logicielle SA , nous considérons un graphe acyclique, orienté et valué $(\mathcal{T}, \mathcal{C})$. L'ensemble des nœuds $\mathcal{T} = \{t_1, \dots, t_n\}$ correspond aux tâches et l'ensemble des arcs $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ représente les messages échangés par les tâches.

Une tâche t_i est définie par ses caractéristiques temporelles et ses besoins en ressource : sa période, T_i (une tâche est périodiquement activée) ; son pire temps d'exécution sans préemption C_i ; et son besoin mémoire, m_i . Les arcs $c_{ij} = (t_i, t_j) \in \mathcal{C}$ sont valués par la quantité de données échangées : d_{ij} . Les tâches qui communiquent ont la même période. De plus, deux types de communications sont envisagés (voir figure 2) :

- communication locale sans délai (voir figure 2.a) : les deux tâches sont sur un même processeur et utilisent la mémoire du processeur pour communiquer ;
- communication distante avec délai (voir figure 2.b) : les tâches sont placées sur des processeurs différents et communiquent via le réseau.

En aucun cas nous ne considérons des contraintes de précérence. Les tâches sont périodiquement activées de manière indépendante : elles lisent et écrivent les données au début et à la fin de leur exécution.

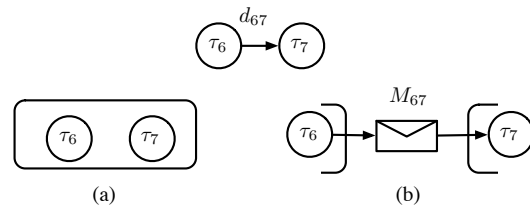


FIG. 2 – Mode de communication entre les tâches : (a) communication locale sans délai, (b) communication distante avec délai

Chaque processeur est ordonnancé par une politique à priorités fixes. L'ordonnement résultant est conduit par les niveaux de priorités et l'ordonnancier décide **en ligne** qu'elle est la tâche à exécuter¹. Le principal problème de l'ordonnement

¹Pour un système temps réel, il existe aussi des ordonnanciers

à priorités fixes n'est donc pas de construire une séquence d'activation des tâches mais de prouver que le système est ordonnançable, c.-à-d. que toutes les tâches respectent leur échéance pour n'importe quelle exécution.

Une priorité, $prio_i$ est attachée à chaque tâche t_i et t_j est dite plus prioritaire que t_i si et seulement si $prio_j > prio_i$ et peut donc la préempter.

2.2 Le problème de placement

Un placement est une application $A : \mathcal{T} \rightarrow \mathcal{P}$ qui place une tâche t_i sur un processeur $p_k : t_i \mapsto A(t_i) = p_k$. Le problème de placement consiste à trouver une application A qui respecte toutes les contraintes décrites ci-après.

2.2.1 Contraintes de ressource

Trois sortes de contraintes de ressource sont considérées :

- **Capacité mémoire** : la mémoire utilisée sur un processeur p_k ne peut pas excéder sa capacité (μ_k) :

$$\forall k = 1..m, \sum_{A(t_i)=p_k} m_i \leq \mu_k \quad (1)$$

- **Facteur d'utilisation** : le facteur d'utilisation d'un processeur ne peut pas excéder sa propre capacité. Le rapport $r_i = C_i/T_i$ signifie qu'un processeur est utilisé à $r_i\%$ par une tâche t_i . L'inégalité suivante est une condition nécessaire pour l'ordonnançabilité :

$$\forall k = 1..m, \sum_{A(t_i)=p_k} \frac{C_i}{T_i} \leq 1 \quad (2)$$

- **Utilisation réseau** : pour éviter une surcharge du réseau, la somme des données transportées ne peut pas excéder la capacité du réseau par unité de temps :

$$\sum_{\substack{c_{ij} = (t_i, t_j) \\ A(t_i) \neq A(t_j)}} \frac{d_{ij}}{T_i} \leq \delta \quad (3)$$

2.2.2 Contraintes de placement

Trois types de contraintes de placement ont été identifiées :

- **Résidence** : une tâche peut avoir besoin d'une ressource logicielle ou matérielle spécifique qui n'est disponible que sur certains processeurs

ceurs cycliques et statiques pour lesquels l'ordonnancement est conduit par le temps et les tâches sont activées suivant une table d'ordonnancement pré-établie.

(par ex., une tâche qui traite l'acquisition d'un capteur doit être présente sur le processeur connecté à l'entrée du périphérique). Une telle contrainte est définie par un couple (t_i, α) où $t_i \in \mathcal{T}$ est une tâche et $\alpha \subseteq \mathcal{P}$ est un ensemble de processeurs sur lesquels sont disponibles la ressource. Un placement A doit respecter :

$$A(t_i) \in \alpha \quad (4)$$

- **Co-résidence** : cette contrainte impose que plusieurs tâches soient placées sur un même processeur (par ex. elles partagent une ressource commune). Cette contrainte est définie par un ensemble de tâches $\beta \subseteq \mathcal{T}$ et doit respecter pour un placement A :

$$\forall (t_i, t_j) \in \beta^2, A(t_i) = A(t_j) \quad (5)$$

- **Exclusion** : certaines tâches doivent être répliquées pour des raisons de tolérance aux fautes et ne doivent pas être assignées sur un même processeur. Cela correspond à un ensemble $\gamma \subseteq \mathcal{T}$ de tâches qui ne doivent pas être placées ensemble. Un placement A doit satisfaire :

$$\forall (t_i, t_j) \in \gamma^2, A(t_i) \neq A(t_j) \quad (6)$$

2.2.3 Contraintes temporelles

Les contraintes temporelles sont exprimées par des échéances implicites. Elles imposent que le temps de réponse d'une tâche t_i , c.-à-d. la durée entre son activation et sa terminaison, soit borné par sa période T_i . Pour valider l'ordonnançabilité, l'idée est de construire le pire scénario d'exécution qui cause le maximum d'interférence sur la tâche t_i . Il suffit alors de calculer le pire temps de réponse R_i qui en résulte et de vérifier la condition $R_i \leq T_i$. Le même raisonnement est fait sur les messages.

Tâches indépendantes. Pour des tâches indépendantes et périodiques sous un ordonnancement préemptif à priorités fixes, il a été prouvé que le pire scénario d'exécution de t_i survient quand toutes les tâches plus prioritaires sont activées simultanément (date d sur la figure 3). Le pire temps de réponse de t_i est alors [9] :

$$R_i = C_i + \sum_{t_j \in hp_i(A)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (7)$$

avec $hp_i(A)$ l'ensemble des tâches plus prioritaires que t_i et placées sur le processeur $A(t_i)$. La résolution de l'équation (7) est discutée dans la partie 3.1.

Architecture matérielle			
\mathcal{P}	Ensemble des processeurs, $ \mathcal{P} = m$	μ_i	Capacité mémoire du processeur p_i
p_i	Processeur i	δ	Capacité du réseau
Architecture logicielle			
\mathcal{T}	Ensemble de tâches, $ \mathcal{T} = n$	\mathcal{C}	Ensemble des messages, $ \mathcal{C} = c$
t_i	Tâche i	c_{ij}	Message de t_i vers t_j
T_i	Période de t_i	T_{ij}	Période de c_{ij} ($T_{ij}=T_i=T_j$)
C_i	Pire temps d'exécution de t_i	C_{ij}	Pire temps de transmission de c_{ij}
R_i	Pire temps de réponse de t_i	R_{ij}	Pire temps de réponse de c_{ij}
m_i	Besoin mémoire de t_i	d_{ij}	Taille de c_{ij}
$prio_i$	Priorité de t_i	$prio_{ij}$	Priorité de c_{ij}

TAB. 1 – Synthèse des paramètres du problème

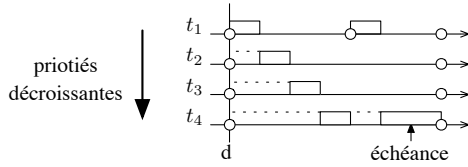


FIG. 3 – Pire scénario d'exécution dans lequel t_4 est préemptée par t_1, t_2, t_3 et t_1 manque son échéance

Message sous CAN. L'ordonnement des messages sur un réseau CAN peut-être perçu comme étant non préemptif à priorités fixes. L'équation (7) peut-être appliquée au message en respectant les différences suivantes : un message ne peut subir que des interférence avant le début de son exécution ; un temps de blocage, c.-à-d. un intervalle de temps pendant lequel un message peut bloquer un autre plus prioritaire, doit être ajouté. Le pire temps de réponse résultant pour un message CAN est [16] :

$$R_{ij} = C_{ij} + L_{ij} \quad (8)$$

$$L_{ij} = \sum_{M_{kl} \in hp_{ij}(A)} \left\lceil \frac{L_{ij} + \tau_{bit}}{T_{kl}} \right\rceil C_{kl} + \max_{M_{kl} \in lp_{ij}(A)} (C_{kl} - \tau_{bit}) \quad (9)$$

où $hp_{ij}(A)$ (respectivement $lp_{ij}(A)$) est l'ensemble des messages présents pour un placement A avec une priorité plus grande (resp. plus petite) que $prio_{ij}$; τ_{bit} est le temps de transmission d'un bit et est lié à la capacité du bus δ . La condition d'ordonnabilité est de nouveau $R_{ij} \leq T_{ij}$. Le temps de transmission C_{ij} dépend de la taille du message, son calcul n'est pas précisé ici et peut-être trouvé dans [16].

3 Résolution

Le problème du placement de tâches temps réel a été étudié depuis longtemps. Des approches in-

complètes et heuristiques telles les algorithmes génétiques [4], la recherche tabou [17] ou le recuit simulé [15] ont déjà été appliquées pour résoudre ce problème. Des approches complètes basées sur la recherche par séparation et évaluation [11] ou sur la programmation par contraintes [14, 3] ont aussi été testées. Cependant, ces travaux divergent du nôtre du point de vue de l'ordonnement : seuls des ordonnements statiques et cycliques ont été étudiés, c.-à-d. que les activations des tâches suivent une table de dates pré-calculées. Le fait d'utiliser ce type d'ordonnement rend le problème très différent du nôtre. Alors que l'ordonnement cyclique est vu comme un intervalle de temps discrets, les contraintes temporelles peuvent être naturellement exprimées par le formalisme classique de la ppc. Le problème principal avec l'ordonnement dynamique à priorités fixes est que la condition d'ordonnabilité n'est pas exprimable sous la forme d'une contrainte classique de ppc.

Nous allons commencer par donner le modèle de contraintes que nous considérons pour ensuite exposer les moyens mis en œuvre pour traiter de la contrainte d'ordonnabilité.

3.1 Le modèle de contraintes

Le modèle de contraintes utilisé est basé sur une formulation redondante de trois ensembles de variables : x , y , et w . Commençons par considérer les n variables x (nos variables de décision) qui correspondent chacune à une tâche et représentent le processeur choisi pour cette tâche : $\forall i \in \{1..n\}$, $x_i \in [1..m]$. Les variables booléennes y indiquent la présence ou non d'une tâche sur un processeur : $\forall i \in \{1..n\}, \forall p \in \{1..m\}$, $y_{ip} \in \{0, 1\}$. Les variables booléennes w sont introduites pour exprimer la présence ou non de deux tâches échangeant un message sur un même processeur : $\forall c_{ij} = (t_i, t_j) \in \mathcal{C}$, $w_{ij} \in \{0, 1\}$. Des contraintes d'intégrité (*channeling constraints*) sont utilisées pour assurer la consistance du modèle. Le tableau 2 donne le modèle de contraintes résultant.

Variables

$$\begin{aligned} x_i &= [1..m], \forall 0 < i \leq n \\ y_{ip} &= \{0, 1\}, \forall 0 < i \leq n, 0 < p \leq m \\ w_{ij} &= \{0, 1\}, \forall 0 < i \leq n, 0 < j \leq n \end{aligned}$$

Contraintes

Résidence	$\forall(i, \alpha), x_i \neq \alpha$
Co-Résidence	$\forall(t_i, t_j) \in \beta^2, x_i = x_j$
Exclusion	$alldifferent(x_i t_i \in \gamma)$
Capacité mémoire	$\forall p \in \{1..m\}, \sum_{i \in \{1..n\}} y_{ip} \times m_i \leq \mu_p$
Facteur d'utilisation	$\forall p \in \{1..m\}, \sum_{i \in \{1..n\}} \frac{ppcm(T) \times C_i \times y_{ip}}{T_i} \leq ppcm(T)$
Utilisation du réseau	$\sum_{i \in \{1..n\}} \frac{ppcm(T) \times d_{ij} \times w_{ij}}{T_i} \leq ppcm(T) \times \delta$
Intégrité du modèle 1	$\forall i, x_i = j \Leftrightarrow y_{ij} = 1$
Intégrité du modèle 2	$\forall c_{ij}, x_i = x_j \Leftrightarrow w_{ij} = 1$

TAB. 2 – Modèle de contraintes (sans celles traitant de l'ordonnançabilité)

Les facteurs d'utilisation et de réseau sont reformulés à l'aide du *ppcm* (plus petit commun multiple) des périodes des tâches car le solveur de contraintes sur lequel nous travaillons ne peut pas dans sa version actuelle prendre en considération des contraintes mixtes, c.-à-d. avec des coefficients réels et des variables entières. Remarquons que la capacité mémoire et le facteur d'utilisation pourraient être mieux intégrés à l'aide d'une contrainte cumulative ou de type *bin-packing*.

Ce modèle ne tient pas compte des contraintes temporelles. Les équations (7) et (8) correspondent à des équations de la forme :

$$x = \sum_{i=1}^k \left\lceil \frac{x}{a_i} \right\rceil b_i + c \quad (10)$$

où x est un entier. En notant $f(x) = \sum_{i=1}^k \left\lceil \frac{x}{a_i} \right\rceil b_i + c$, ce problème consiste à trouver le point fixe x^* de f tel que $f(x^*) = x^*$. À cause de la fonction partie entière la résolution n'est pas si simple. Il est facile de montrer que le point fixe peut-être atteint en un nombre fini de pas mais avec une complexité pseudo-polynomiale. À notre connaissance, la complexité de cette équation n'est pas connue [13]. La contrainte temporelle est donc difficile à exprimer par une contrainte primitive du solveur. Les parties suivantes s'attachent à décrire un moyen pour intégrer les contraintes temporelles dans une contrainte globale.

3.2 Une contrainte globale pour l'ordonnançabilité

Nous proposons dans cette partie, une contrainte globale dédiée aux aspects temporels et qui prend

en considération la résolution des équations définies précédemment.

3.2.1 Sémantique opérationnelle

La contrainte *schedulable* est définie par trois paramètres : un ensemble $X = \{x_1, \dots, x_n\}$ de variables, une architecture matérielle $HA = (\mathcal{P}, \mathcal{N})$ et une architecture logicielle $SA = (\mathcal{T}, \mathcal{C})$. Nous avons $|\mathcal{T}| = n$ et $\forall i, x_i \in [1, \dots, |\mathcal{P}|]$. La sémantique résultant de la contrainte est :

Définition 3.1 *schedulable*(X, HA, SA) est vérifiée, si et seulement si, pour X un placement des tâches de SA sur HA :

$$\forall t_i \in \mathcal{T} \quad R_i \leq T_i \quad \wedge \quad \forall c_{ij} \in \mathcal{C} \quad R_{ij} \leq T_{ij} \quad (11)$$

R_i (resp. R_{ij}) est le pire temps de réponse de la tâche t_i (resp. du message entre t_i et t_j) défini par les équations (7) (resp. équation (8)).

En d'autres termes, la contrainte est vérifiée, si et seulement si, le placement X est ordonnançable.

3.2.2 Complexité

Nous considérons par la suite la contrainte *schedulable* dans le cadre spécifique d'un réseau CAN en mode désynchronisé. Un premier résultat important peut-être obtenu pour cette contrainte :

Propriété 3.1 Assurer l'arc-consistance pour la contrainte *schedulable* est NP-dur.

Preuve : [10] montre à l'aide d'une réduction par 3-PARTITION, que vérifier l'ordonnançabilité est NP-dur. De plus, comme la classe de complexité

des équations (10) (7) et (8) est encore inconnue [13], son appartenance à NP n'est pas encore prouvée, d'où le résultat. \square

Notre algorithme de filtrage est donc basé sur une forme plus faible de consistance obtenue à l'aide de certains raisonnements hypothétiques.

3.2.3 Filtrage

Le filtrage est essentiellement incrémental. Partant d'un placement partiel qui est ordonnançable, nous prenons en considération le fait que placer une tâche sur un processeur (instanciation de variable) permet d'interdire le placement de certaines tâches sur des processeurs (retrait de valeur).

Plus formellement, un placement partiel est une application $a : \mathcal{U} \subset \mathcal{T} \rightarrow \mathcal{P} \cup \{-1\}$ telle que

$$\begin{aligned} t_i \in \mathcal{U} &\mapsto a(t_i) = p_k \\ t_i \in \mathcal{T} - \mathcal{U} &\mapsto a(t_i) = -1 \end{aligned} \quad (12)$$

Une extension a' d'un placement partiel a par l'affectation d'une tâche t_i sur un processeur p ($x_i = p$) telle que $a(t_i) = -1$ est définie par $\forall t_k \in \mathcal{U}, a'(t_k) = a(t_k)$ et $a'(t_i) = p$. Elle est notée $a' \leftarrow a + (x_i = p)$. De la même manière, $a + (x_i, x_j) = p$ désigne une extension d'un placement partiel a par l'allocation de deux tâches t_i et t_j sur le même processeur ($x_i = p$ et $x_j = p$).

Pour comprendre le fonctionnement incrémental de notre algorithme de filtrage, il est important de garder en mémoire les remarques suivantes (variables avec CAN) :

- en considérant un placement partiel ordonnançable a et son extension $a' \leftarrow a + (x_i = p)$, seule l'ordonnançabilité des tâches placées sur p qui ont une priorité inférieure ou égale à celle de t_i doit être testée ;
- la présence d'un nouveau message M_{ab} ne remet en question que l'ordonnançabilité des messages qui ont une plus petite priorité que M_{ab} et tous ceux avec une priorité plus grande tels que $C_{ab} > \max_{M_{kl} \in \mathcal{P}_{i,j}(a)} C_{kl}$.

Le cœur de l'algorithme de filtrage repose sur la méthode $\text{sched}(a, a')$ (respectivement $\text{schedM}(mes, mes')$) qui calcule le point fixe de l'équation (7) (resp. l'équation (8)) pour prouver que le placement a' (resp. l'ensemble des messages mes') est ordonnançable en tenant compte du fait que a' (resp. mes') est une extension de a (resp. mes) que nous savons ordonnançable.

Les informations pouvant être apprises lors d'une intanciation de variable (nouveau placement) ou lors du retrait d'une valeur (une tâche est interdite sur un processeur) sont de plusieurs types. Par

exemple, quand un message entre deux tâches (si elles ne sont pas sur un même processeur) est non ordonnançable alors les deux tâches doivent être placées sur un même processeur. Cette information doit être maintenue.

Les algorithmes de la figure 4 donnent le comportement de la contrainte quand une tâche est allouée sur un processeur ($x_i = p$ — méthode $\text{awakeOnInst}(x_i = p)$) et quand une valeur est retirée ($x_i \neq p$ — méthode $\text{awakeOnRem}(x_i \neq p)$).

Les informations suivantes sont nécessaires et doivent être maintenues lors de la résolution :

- le placement partiel courant a ;
- l'ensemble des messages mes qui transitent sur le réseau (mes peut contenir un message c_{ij} pour lequel t_i et t_j ne font pas partie de a mais pour lesquels les domaines sont disjoints) ;
- la liste de tâches $link_i$ qui doivent être placées avec t_i pour éviter la non-ordonnançabilité du réseau. L'invariant suivant est maintenu par la contrainte : $\forall x_k \in link_i, x_k = x_i$.

L'algorithme commence par mettre à jour les structures de données a , $link_i$ et mes et vérifier l'ordonnançabilité du nouveau placement partiel a (ligne 4 de UpdateAllocation) ou de l'ensemble des messages (ligne 3 de UpdateNetwork). Si aucune contradiction ne se produit, le filtrage peut commencer :

- Si une tâche a été ajoutée à a , l'ordonnançabilité de toutes les autres tâches est vérifiée en tenant compte des relations imposées par $link$ (UpdateAllocation) ;
- Si un nouveau message a été ajouté à mes , l'ordonnançabilité des messages est vérifiée (UpdateNetwork). Si un message est non-ordonnançable avec mes , les deux tâches correspondantes doivent être placées ensemble, ainsi la structure de données $link$ est mise à jour (ligne 9–10). Puis l'ordonnançabilité du placement simultané des deux tâches est vérifiée sur chacun des processeurs de leur domaine (lignes 11–14).

Propriété 3.2 *Les algorithmes de la figure 4 assurent que :*

- le placement partiel défini par les tâches instanciées est ordonnançable ;
- chaque extension de a par le placement d'une tâche t_i sur un processeur p ($x_i = p$) tel que $p \in D_i$, est ordonnançable.

Preuve : Une fois le point fixe atteint, a et mes sont ordonnançables. Une extension de a avec $x_i = p$ pourrait ne pas être ordonnançable pour deux raisons : x_i n'est pas ordonnançable avec l'ensemble des tâches placées sur p (lignes 6–12 de

Procedure: awakeOnInst($x_i = p$)

```

1: if ( $a(i) = -1$ )
2:   for all ( $x_k \in link_i$ ) do
3:     instantiate  $x_k$  to  $p$ ;
4:   UpdateNetwork( $x_i$ );
5:   UpdateAllocation( $x_i = p$ );

```

Procedure: awakeOnRem($x_i \neq p$)

```

1: for all ( $x_k \in link_i$ ) do
2:   remove  $p$  from  $D_k$ 
3:   UpdateNetwork( $x_i$ );

```

Procedure: UpdateAllocation($x_i = p$)

```

1:  $a_1 \leftarrow a + (x_i = p)$ ;
2: for all ( $x_k \in link_i$ ) do
3:    $a_1 \leftarrow a_1 + (x_k = p)$ ;
4: if (!sched( $a, a_1$ )) fail;
5:  $a \leftarrow a_1$ ;
6: for all  $x_j$  s.t.  $|D_j| > 1$  and  $p \in D_j$  do
7:   if ( $link_j \neq \emptyset$ )
8:     for all ( $x_k \in link_j$  s.t.  $k > j$ ) do
9:       if (!sched( $a, a + (x_j, x_k) = p$ ))
10:        remove  $p$  from  $D_k, D_j$ ;
11:   else if (!sched( $a, a + (x_j = p)$ ))
12:     remove  $p$  from  $D_j$ ;

```

Procedure: UpdateNetwork(x_i)

```

1:  $mesChanged \leftarrow false$ ;
2: for all  $c_{ij} \notin mes$  s.t.  $D_i \cap D_j = \emptyset$ 
3:   if (!schedM( $mes, mes \cup \{c_{ij}\}$ )) fail;
4:    $mes \leftarrow mes \cup \{c_{ij}\}$ ;
5:    $mesChanged \leftarrow true$ ;
6: if ( $mesChanged$ )
7:   for all ( $c_{kl} \in \mathcal{C}$  s.t.  $c_{kl} \notin mes$ ) do
8:     if !schedM( $mes, mes \cup \{c_{kl}\}$ )
9:        $link_k \leftarrow link_k \cup \{x_l\}$ ;
10:       $link_l \leftarrow link_l \cup \{x_k\}$ ;
11:       $D_k, D_l \leftarrow D_k \cap D_l$ ;
12:      for all  $p \in D_k$ ;
13:        if (!sched( $a, a + (x_l, x_k) = p$ ))
14:          remove  $p$  from  $D_k, D_l$ ;

```

FIG. 4 – L’algorithme *UpdateAllocation* décrit le filtrage suite à un placement x_i ($x_i = p$). L’algorithme *UpdateNetwork* est appelé quand le domaine de x_i est modifié, ce qui peut modifier l’ensemble des messages.

UpdateAllocation, ce qui ne peut pas être le cas si $p \in D_i$); le placement de x_i donne naissance à un nouveau message non ordonnançable (les traitements des lignes 7–14 de *UpdateNetwork* empêchent ce cas de se produire). \square

La complexité de l’algorithme de filtrage est en $O(n + c \times m)$. Pour réduire le nombre d’appels à *sched*(a, a') et ainsi réduire le nombre de résolutions de l’équation (10), il est possible de précalculer des règles de dominance entre les tâches :

Propriété 3.3 Si $x_i = p_k$ rend t_i non-ordonnançable, alors p_k peut être supprimé du domaine D_j des tâches t_j telles que $prio_j < prio_i \wedge C_j \geq C_i \wedge T_j \leq T_i$.

Propriété 3.4 Si $x_i = p_k$ rend t_b non-ordonnançable, alors p_k peut être supprimé du domaine D_j des tâches t_j telles que $prio_j > prio_b \wedge C_j \geq C_i \wedge T_j \leq T_i$.

Preuves : ces propriétés sont immédiatement prouvées par récurrence sur les valeurs de R_j^k ou R_b^k à chaque itération de l’algorithme de calcul du point fixe. \square

4 Résultats expérimentaux

Dans un travail précédent [2], nous avons résolu le problème de placement à l’aide d’une approche utilisant la décomposition de Benders [1] et plus spécifiquement une décomposition basée sur la logique [6]. Nous ne rappellerons ici que les principaux concepts développés dans [2].

Le problème de placement est décomposé en deux, avec d’un côté un problème maître traitant des contraintes de placement et de ressource, et de l’autre un problème esclave associé aux contraintes d’ordonnançabilité. Le problème maître est résolu par la ppc et produit un placement qui satisfait seulement les contraintes de placement et de ressource. Le problème esclave vérifie l’ordonnançabilité et produit un ensemble de contraintes (symboliques et arithmétiques) qui exclue tous les placements qui seraient non-ordonnançables pour des raisons similaires. Cette approche conserve l’élément central de la décomposition de Benders : les coupes de Benders. La vitesse de convergence et le succès de la technique dépendent fortement de la qualité des coupes. De plus, le problème maître utilise un mécanisme de retour-arrière pour éviter les calculs redondants quand une coupe est ajoutée.

Ici, nous proposons de comparer l’approche par décomposition de Benders avec la nouvelle contrainte globale.

Mémoire		Placement			Ordonnabilité		Réseau			
	% <i>mem</i>		% <i>res</i>	% <i>co</i>	% <i>exc</i>		% <i>global</i>		<i>mes</i>	% <i>mes</i>
1	60	1	0	0	0	1	40	1	0	0
2	30	2	15	15	15	2	60	2	20	70
3	10	3	33	33	33	3	90	3	30	150

TAB. 3 – Les classes de difficulté

Pour le problème de placement il n'existe aucun jeu d'essais issu de la communauté temps réel. Les expérimentations sont habituellement faites sur des exemples didactiques ou à partir de configurations générées aléatoirement. Nous avons choisi la dernière solution et utilisé le même générateur que dans [2]. Il est basé sur les principes suivants :

- **Tâches.** le nombre de tâches n est fixé à 40. Les périodes sont générées de manière à limiter la valeur de $ppcm(T)$ en conservant un maximum de valeurs [5]. Les priorités sont affectées aléatoirement. Un paramètre, %*global*, est utilisé pour régler le facteur d'utilisation global tel que $\sum_{i=1}^n C_i/T_i = m\%_{global}$. La difficulté du problème d'ordonnabilité dépend de %*global* pour lequel les valeurs varient de 40 à 90.
- **Communications.** Le nombre de communications, c , c.-à-d. le nombre d'arcs dans le graphe des tâches, et la charge potentielle du réseau %*mes*, ont un impact sur la difficulté du problème. Pour une raison de simplicité, seules les communications linéaires entre les tâches sont considérées et la priorité des messages est la même que celle de la tâche productrice. Les pires temps de transmission sont générés tels que $\sum_{i=1}^n C_{ij}/T_i = \%_{mes}$.
- **Processeurs.** Le nombre de processeurs, m , est fixé à 7. La capacité mémoire d'un processeur est générée telle que $\sum_{k=1}^m m_k = (1 + \%_{mem}) \sum_{i=1}^n \mu_i$ où %*mem* représente la capacité mémoire supplémentaire disponible sur l'architecture matérielle.
- **Contraintes de placement.** Le pourcentage de tâches participant à une contrainte de placement est donné par le paramètre %*res* pour les contraintes de résidence, %*co* pour celles de co-résidence, et %*exc* pour celles d'exclusion. Les différentes contraintes sont dimensionnées de manière à ce que %*res*, %*co* et %*exc* soient respectés. La difficulté d'un problème est donnée en fonction de ces différents pourcentages.

Plusieurs classes de problèmes ont été définies en fonction des contraintes. Le tableau 3 décrit les paramètres utilisés pour chaque classe. En combinant ces paramètres, plusieurs catégories ont été définies. Une catégorie W-X-Y-Z correspond à un problème avec une difficulté de classe W pour la mé-

moire, une classe X pour les contraintes de placement et de ressource, une classe Y pour l'ordonnabilité et une classe Z pour le réseau.

Le tableau 4 présente les résultats obtenus avec Benders et avec la contrainte globale : %Suc donne le pourcentage d'instances résolues ; NbC le nombre d'instances consistantes ; Time(s) le temps en seconde de résolution ; Nodes le nombre de nœuds explorés ; Iter le nombre d'itérations de la méthode de Benders ; Noe le nombre de contraintes de type *NotAllequal* et Comb le nombre de combinaisons linéaires correspondant aux coupes extraites de la méthode de Benders. Les valeurs moyennes de ces données sont présentées ici pour les instances résolues (une solution est trouvée ou l'inconsistance est prouvée) sur 100 instances par catégorie. Une limite de temps a été fixée à 10 minutes par instance. La stratégie de recherche utilisée est basée sur une approche par impacts [12].

Le tableau 5 détaille le temps de résolution et donne les valeurs moyennes, médianes, minimales et maximales pour les instances résolues.

Les résultats montrent que l'approche basée sur la contrainte globale (97.8% d'instances résolues) est très compétitive par rapport à celle de Benders (97.3% d'instances résolues). Les valeurs minimales et maximales des temps de résolution montrent que Benders peut-être plus efficace pour les problèmes où l'ordonnement est facile (classe 2-2-2-1 où le temps moyen avec Benders est nettement inférieur à celui avec la contrainte globale). Cela s'explique par le fait que la contrainte globale a à résoudre beaucoup d'équations d'ordonnabilité qui ont peu d'impact sur l'élagage alors que Benders ne considère que les contraintes de placement et de ressource.

Pour les catégories de problèmes difficiles en ordonnancement (par ex. la classe 1-1-3-1) et celles avec une occupation réseau importante (par ex. les classes 1-2-2-3 et 2-2-2-3), Benders est clairement moins rapide que la contrainte globale. Cela s'explique par le fait que pour Benders les contraintes liées à l'ordonnabilité sont entièrement apprises au cours de la résolution et donc peu efficace pour guider la recherche au début. La contrainte globale permet d'améliorer ce comportement. Pour une charge réseau importante, Benders a la même difficulté pour tenir compte de l'ordonnabilité

	Benders							CP			
	%Suc	NbC	Time(s)	Nodes	Iter	Noe	Cut	%Suc	NbC	Time(s)	Nodes
1-1-3-1	99	99	5,6	3105,9	107,5	482,5	NA	99	99	3,8	1685,3
1-2-2-3	99	76	15,5	7227,4	629,6	192,8	1853,2	100	76	5,1	1085,8
2-2-2-1	100	56	0,4	370,6	19,9	79,7	NA	100	56	5,3	1985,9
2-2-2-2	100	70	0,3	282,7	17,2	48,5	12,5	100	70	0,3	36,6
2-2-2-3	92	70	28,2	10276,8	662	262,2	1784,9	95	73	13,3	4323,3
2-2-3-1	90	30	8,3	3267,8	30,1	154,2	NA	90	30	10,4	4059,8
2-3-2-1	100	19	0,4	172,6	6,8	37,3	NA	100	19	0,3	9,2
3-2-2-1	99	57	1,4	960,2	39,6	143,2	NA	99	57	1,4	458,1

TAB. 4 – Moyennes des résultats pour chaque approche.

	Benders					CP				
	%Suc	Time(s)				%Suc	Time(s)			
	Av	Med	Min	Max		Av	Med	Min	Max	
1-1-3-1	99	5,6	2,8	0,27	60,6	99	3,8	0,45	0,3	78,6
1-2-2-3	99	15,5	2,6	0,19	519,8	100	5,1	0,58	0,23	336,2
2-2-2-1	100	0,4	0,27	0,17	4,2	100	5,3	0,34	0,23	314,9
2-2-2-2	100	0,3	0,29	0	2,6	100	0,3	0,31	0	4,1
2-2-2-3	92	28,2	4,7	0	386,9	95	13,3	0,73	0	403,4
2-2-3-1	90	8,3	0,25	0,17	344	90	10,4	0,36	0	375
2-3-2-1	100	0,4	0,26	0,17	6,5	100	0,3	0,37	0	1,9
3-2-2-1	99	1,4	0,32	0,17	15	99	1,4	0,33	0,23	91,2

TAB. 5 – Valeurs Moyennes, médianes, minimums et maximums des temps de résolution.

des messages.

Le résultat le plus significatif est qu'en pratique l'aspect pseudo-polynomial des équations d'ordonnancement n'est pas un facteur bloquant pour l'élagage. Ainsi, la contrainte globale est efficace pour ce genre de problème.

5 Conclusion

Dans ce papier, une nouvelle approche a été définie à l'aide d'une contrainte globale pour résoudre les problèmes de placement des systèmes temps réel durs. La difficulté et la nouveauté de ce problème repose dans les contraintes d'ordonnement propre à ce genre de système. Nous proposons une contrainte globale dédiée pour embarquer la résolution des équations servant à prouver l'ordonnancement du système. Cependant, ces équations sont connues comme étant d'une complexité pseudo-polynomiale, ce qui rend NP-dur la maintenance de l'arc consistant. Notre résultat principal a été de montrer qu'en pratique ces équations peuvent être résolues à chaque nœud de l'arbre de recherche, rendant cette approche performante pour ce problème.

Références

- [1] J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4 :238–252, 1962.
- [2] Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jusien, and Yvon Trinquet. Decomposition and learning for a real-time task allocation problem. In *CP 2004*, pages 153–167, 2004.
- [3] C. Ekelin. *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. PhD thesis, Chalmers University of Technology, 2004.
- [4] E. Ferro, R. Cayssials, and J. Orozco. Tuning the Cost Function in a Genetic/Heuristic Approach to the Hard Real-Time Multiprocessor Assignment Problem. *Proceeding of the Third World Multiconference on Systemics Cybernetics and Informatics*, pages 143–147, 1999.
- [5] J. Goossens and C. Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the RTS Embedded System conference (RTS'01)*, pages 133–147, 2001.
- [6] J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96 :33–60, 2003.
- [7] Ulrich Junker. Quickxplain : Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, Seattle, WA, USA, August 2001.
- [8] E. L. Lawler. Recent Results in the Theory of Machine Scheduling. *Mathematical Programming : The State of the Art*, pages 202–233, 1983.

- [9] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines, In *11th IEEE Real-Time Systems Symposium (RTSS 1990)*, pages 201–209, 1990
- [10] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2 :237–250, 1982.
- [11] M. Mutka and J-P. Li. A tool for allocating periodic real-time tasks to a set of processors. *The Journal of Systems and Software*, 29(2) :135–164, 1995.
- [12] P. Refalo. Impact-based search strategies for constraint programming. In *CP 2004*, pages 556–571, 2004.
- [13] P. Richard. Polynomial-time approximate schedulability tests for fixed-priority real-time tasks : Some numerical experimentations. In *14th Int. Conf. on Real-Time and Network Systems*, pages 191–199, 2006.
- [14] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4) :335–357, 2000.
- [15] K. Tindell, A. Burns, and A. Wellings. Allocation Hard Real-Time tasks : An NP-Hard Problem Made Easy. *The Journal of Real-Time Systems*, 4(2) :145–165, 1992.
- [16] K. Tindell, H. Hansson, and A. Wellings. Analysis of real-time communications : controller area network (can). In *15th IEEE Real-Time Systems Symposium (RTSS 1994)*, pages 259–265, 1994.
- [17] L. Vargas and R. Olivaira. Empirical study of tabu search, simulated annealing and multi-start in field bus scheduling. In *IEEE Conference on Emerging Technologies and Factory Automation ETFA*, volume 1, pages 101–108, 2005.