

Décomposition et apprentissage pour un problème d'allocation de tâches temps réel

Hadrien Cambazard*, Pierre-Emmanuel Hladik**,
Anne-Marie Déplanche**, Narendra Jussien*, Yvon Trinquet**
*École des Mines de Nantes
LINA/FRE CNRS 2729
4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3, France
{hcambaza,jussien}@emn.fr
**IRCCyN
UMR CNRS 6597
1 rue de la Noë – BP 92101
F-44321 Nantes Cedex 3, France
{hladik,deplanche,trinquet}@irccyn.ec-nantes.fr

Résumé

La décomposition de Benders a été utilisée avec succès pour de nombreuses problématiques en Recherche Opérationnelle. Nous présentons ici la mise en œuvre d'une technique de coopération fondée sur une généralisation du cadre classique de la décomposition de Benders et appliquée à la résolution d'un problème d'allocation de tâches temps réel dans un système réparti (ordonnancement préemptif à priorités fixes pour des tâches périodiques). Un problème maître, résolu par programmation par contraintes coopère avec un sous-problème analytique. Celui-ci est traité par des techniques issues des travaux de la communauté temps réel et couplées avec un algorithme *ad hoc* de détection de conflits. Les contraintes et nogoods appris au cours de la recherche jouent un rôle similaire aux coupes de Benders.

1 Introduction

Les problématiques temps réel interviennent au cœur de systèmes embarqués dans de nombreux domaines : automobile, robotique, aéronautique, télécommunications, ... Les applications actuelles (ex : les voitures) mettent en œuvre un ensemble de calculateurs spécialisés (ex : contrôle moteur, ABS, contrôle de châssis, climatisation, ...) qui

reçoivent des données de capteurs, les traitent et calculent des commandes adéquates, qu'ils envoient vers des actionneurs. Leur caractéristique centrale réside dans le respect d'exigences fonctionnelles mais aussi extra-fonctionnelles telles que les contraintes temporelles. Un système temps réel *dur* est un système pour lequel le non-respect d'une échéance peut causer des défaillances graves et mettre en danger tout le système. Nous nous intéressons aux systèmes temps réel durs distribués (les processeurs sont interconnectés par un réseau de terrain) pour lesquels les unités de traitement ou processeurs respectent une politique d'ordonnancement préemptif à priorités fixes. Les applications considérées sont implémentées par des tâches périodiques, communicantes et dont les temps de réponse peuvent être contraints. Dans ce domaine, de nombreux travaux se fondent sur des analyses d'ordonnancabilité (initiées par Liu et Layland [LL73]) et l'une des principales techniques s'appuie sur l'évaluation du pire scénario d'exécution. Elles ont ensuite connu de nombreuses extensions comme la prise en compte des ressources partagées, des systèmes distribués [TC94] ou des liens de précédence [HKL91]. Le problème que nous abordons est plus particulièrement celui du placement des tâches temps réel dures, c'est-à-dire l'allocation des tâches sur les différents processeurs du système respectant l'ordonnancabilité. Un tel problème a été montré NP-difficile [Law83]. Des solutions ont déjà été proposées par des méthodes heuristiques [FCO99, Ram90], de recuit simulé [TBW92, BM94] et d'algorithmes génétiques [FCO99, San96]. Néanmoins, ces techniques sont souvent incomplètes et peuvent ne pas trouver de placement après des temps de calculs importants. Ainsi, de nouvelles méthodes sont toujours envisagées.

Notre approche s'appuie sur les principes de la décomposition de Benders en découplant le problème du placement proprement dit de celui de l'ordonnancabilité. D'un côté, la programmation par contraintes offre des outils performants pour traiter cette problématique d'affectation, de l'autre, les techniques temps réel sont capables d'analyser avec précision l'ordonnancabilité d'un système. La technique s'inscrit dans le cadre des schémas d'hybridation exploitant les atouts de la programmation par contraintes et ceux des techniques d'optimisation utilisées en recherche opérationnelle [Tho01, JG01, HOTK00, BGR02].

La deuxième partie de cet article introduira le problème et ses paramètres. La présentation de la stratégie de résolution, du cadre de la décomposition de Benders dans lequel elle s'inscrit ainsi que des travaux connexes feront l'objet de la troisième partie. Nous aborderons ensuite la mise en œuvre de la résolution à travers la description des problèmes maître et esclave ainsi que la coopération entre eux par nogoods, avant de conclure par les résultats expérimentaux obtenus.

2 Description du problème

Nous introduisons dans cette section les données et la nature du problème en mettant celui-ci en perspective par rapport à l'ordonnancement hors ligne.

2.1 L'architecture du système temps réel

Le système temps réel est ici modélisé sous la forme d'une architecture logicielle (l'ensemble des tâches applicatives) et d'une architecture matérielle (le support physique d'exécution des tâches). Ce modèle est celui utilisé par Tindell [TBW92].

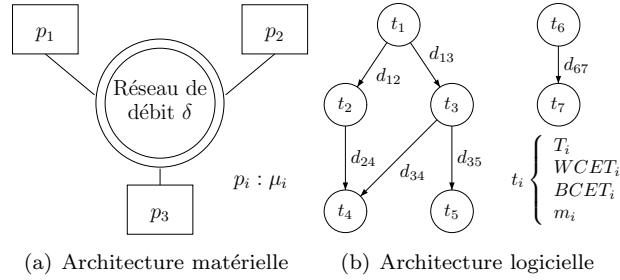


FIG. 1 – Synthèse des paramètres du problème

2.1.1 L'architecture matérielle

L'**architecture matérielle** est constituée d'un ensemble $\mathcal{P} = \{p_1, \dots, p_k, \dots, p_m\}$ de m processeurs de capacité mémoire μ_k s'échangeant des informations sur un réseau. Les processeurs sont considérés identiques du point de vue du traitement (le temps d'exécution d'une tâche est le même quel que soit le processeur).

Les processeurs communiquent via un réseau caractérisé par son débit δ (la quantité d'information transmissible par unité de temps) et son protocole. Un protocole utilisé par les réseaux de terrain est celui de l'anneau à jeton (du fait de son caractère déterministe) : un processeur peut transmettre des données sur le réseau s'il est en possession du jeton, et cela seulement pendant une durée donnée. Après cette durée, ou si le processeur n'a plus de données à envoyer, le jeton est passé au processeur suivant de l'anneau. La durée maximale de présence du jeton sur un processeur doit garantir un temps suffisant pour envoyer tous les messages en attente sur les processeurs.

2.1.2 L'architecture logicielle

Le modèle que nous employons pour décrire l'**architecture logicielle** se compose d'un graphe $(\mathcal{T}, \mathcal{C})$ orienté, acyclique et valué, pour lequel l'ensemble des nœuds $\mathcal{T} = \{t_1, \dots, t_n\}$ désigne les *tâches* (ou *processus*) et l'ensemble des arcs, $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$, représente les communications entre tâches. Un arc $c_{ij} = (t_i, t_j) \in \mathcal{C}$ modélise l'envoi d'un message de t_i vers t_j . Une tâche t_i est définie par : sa période, T_i , son temps maximum d'exécution sans préemption, $WCET_i$ et son besoin en mémoire m_i ¹. Les arcs $c_{ij} = (t_i, t_j) \in \mathcal{C}$ sont caractérisés par la quantité d_{ij} d'information échangée. Les tâches peuvent communiquer en utilisant le réseau et son protocole ou la mémoire locale du processeur. Dans ce dernier cas, la communication s'effectue sans délai mais les tâches doivent être situées sur le même processeur. Dans aucun cas nous ne considérons des liens de précedence. Les tâches sont activées périodiquement de façon indépendante. Elles lisent et écrivent des données au début et à la fin de leur exécution. Enfin, plusieurs grandes catégories d'ordonnancement existent. Nous considérerons que tous les processeurs sont munis d'un ordonnanceur préemptif à priorités fixes. Un niveau de priorité unique $prio_i$ est affecté à chaque tâche et une tâche t_i est dite plus prioritaire que t_j ssi $prio_i > prio_j$. L'exécution d'une instance de tâche peut être préemptée (suspendue) jusqu'à ce que toutes les instances de tâches plus prioritaires prêtes, terminent leur exécution.

¹Comme pour les temps d'exécution, nous ferons l'hypothèse que le besoin mémoire est indépendant du processeur.

2.2 Le problème de placement/ordonnancement

Un placement ou une allocation est une application A qui, à une tâche t_i de \mathcal{T} , associe un processeur p_k de \mathcal{P} :

$$\begin{aligned} A : \mathcal{T} &\rightarrow \mathcal{P} \\ t_i &\mapsto A(t_i) = p_k \end{aligned} \quad (1)$$

Le problème de placement consiste donc à trouver l'application A respectant l'ensemble des contraintes ci-dessous.

2.2.1 Contraintes temporelles

Les contraintes temporelles s'expriment par des **échéances** portant sur la terminaison des tâches. Ces échéances doivent assurer que la durée séparant la date d'activation de toute instance d'une tâche t_i et la fin de transmission des données émises par celle-ci (ou la fin de son exécution si elle ne produit pas de données) doit être inférieure à sa période T_i . Notons TRT le plus grand délai possible d'émission des données sur le réseau, ainsi pour une tâche t_i , son échéance D_i (les contraintes elle-même sont explicitées au 4.2.1) est égale à :

- $T_i - TRT$ si t_i émet des données vers une tâche allouée sur un autre processeur ;
- T_i sinon.

2.2.2 Contraintes de ressources

Trois types de ressources sont considérées : mémoire des processeurs, utilisation des processeurs et utilisation du réseau.

- **Capacité mémoire**² : la somme des besoins mémoire de l'ensemble des tâches résidant sur un processeur p_k ne doit pas excéder la capacité (μ_k) du dit processeur :

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

- **Facteur d'utilisation** : pour que toutes les tâches puissent s'exécuter il est nécessaire que le taux d'utilisation des processeurs ne soit pas supérieur à leur capacité de traitement. Le ratio $r_i = WCET_i/T_i$ signifie qu'un processeur est utilisé r_i pourcent du temps par la tâche t_i . Une première condition nécessaire d'ordonnancement s'écrit :

$$\forall k = 1..m, \quad \sum_{A(t_i)=p_k} \frac{WCET_i}{T_i} \leq 1 \quad (3)$$

- **Utilisation du réseau** : pour que tous les messages puissent être acheminés sur le réseau il est nécessaire que la somme des quantités de données transportées par le réseau par unité de temps soit inférieure au débit du réseau :

$$\sum_{\substack{c_{ij} = (t_i, t_j) \\ A(t_i) \neq A(t_j)}} \frac{d_{ij}}{T_i} \leq \delta \quad (4)$$

²Il serait possible, sans nuire à la généralité du problème traité, de différencier la mémoire en différents types (RAM, ROM, EEPROM ...), chaque type ayant une taille propre pour un processeur donné.

2.2.3 Contraintes de placement

Les contraintes dites de placement concernent l'affectation des tâches sur l'ensemble des processeurs. Trois types de contraintes interviennent sur le placement : résidence, co-résidence et exclusion.

- **Résidence** : cette contrainte est due à la nécessité pour une tâche d'utiliser des ressources logicielles ou matérielles particulières mises à disposition par certains processeurs (ex : un capteur de vitesse couplé avec un dispositif matériel présent sur un ordinateur donné). Il s'agit d'un couple (t_i, α) où $t_i \in \mathcal{T}$ est une tâche et $\alpha \subseteq \mathcal{P}$ est l'ensemble de processeurs sur lesquels la tâche peut s'exécuter. Un placement donné A doit alors vérifier :

$$A(t_i) \in \alpha \quad (5)$$

- **Co-résidence** : cette contrainte stipule que plusieurs tâches doivent être placées sur le même processeur (ex : ces tâches nécessitent le partage d'une ressource commune). Une contrainte de co-résidence est définie par un ensemble de tâches $\beta \subseteq \mathcal{T}$ et tout placement A doit satisfaire :

$$\forall (t_i, t_j) \in \beta^2, A(t_i) = A(t_j) \quad (6)$$

- **Exclusion** : il s'agit d'interdire qu'un ensemble de tâches résident sur les mêmes processeurs. C'est en particulier le cas lorsque des tâches sont des répliques redondantes d'autres tâches à des fins de tolérance aux fautes. Il s'agit d'un ensemble $\gamma \subseteq \mathcal{T}$ de tâches et tout placement A doit satisfaire :

$$\forall (t_i, t_j) \in \gamma^2, A(t_i) \neq A(t_j) \quad (7)$$

Une allocation est dite **valide** si elle vérifie les contraintes de placement et de ressources. Elle est dite **ordonnançable** si elle vérifie les contraintes temporelles. Une allocation valide et ordonnançable est une solution du problème.

2.3 Ordonnancement temps réel et ordonnancement hors ligne

Les problèmes d'ordonnancement sont souvent caractérisés à travers des ensembles de tâches (*jobs*), de processeurs (*machines*) et de ressources additionnelles. Le problème consiste à trouver une affectation des tâches aux processeurs ainsi qu'une date de démarrage pour l'exécution de chaque tâche. Une solution s'apparente donc à un diagramme de Gantt. Dans un contexte temps réel préemptif, on ne peut pas fixer à l'avance la date d'exécution d'une tâche. On ne peut pas non plus violer une échéance car ce sont des processus critiques du système (c'est un problème de satisfaction). Il s'agit cependant d'un problème de décision hors-ligne et on doit fournir une affectation des tâches aux processeurs qui soit ordonnançable en toute circonstance.

3 Approches par décomposition

Notre approche s'appuie dans une large mesure sur la décomposition de Benders [Ben62] qui met en œuvre une stratégie de résolution fondée sur le partitionnement du

problème selon ses variables : (x, y) . On peut la voir comme une forme d'apprentissage par l'échec. Elle s'utilise sur les problèmes ayant la forme suivante :

$$\begin{aligned} \text{P : Min } & f(x) + cy \\ \text{s.t : } & g(x) + Ay \geq a \text{ with : } x \in D, y \geq 0 \end{aligned}$$

Un problème maître (PM) est utilisé pour traiter un sous-ensemble de variables x (D dénote un domaine discret car ce sont souvent les variables entières), puis un problème esclave (PE) complète l'affectation sur y . Si cette affectation est impossible, PE produit une coupe dite de Benders ajoutée au problème maître. La coupe de la forme $z \geq h(x)$ est l'élément clef du processus, elle est fournie par le problème esclave et la résolution de son dual. Considérant une affectation x^* donnée par le maître, le problème esclave (PE) s'écrit avec son dual (PED) :

$$\begin{array}{ll} \text{PE : Min } cy & \text{PED : Max } u(a - g(x^*)) \\ \text{Tel que : } Ay \geq a - g(x^*) \text{ with : } y \geq 0 & \text{Tel que : } uA \leq c \text{ with : } u \geq 0 \end{array}$$

La dualité nous assure que $cy \geq u(a - g(x^*))$, $u(a - g(x^*))$ est donc une borne inférieure de cy . Comme la faisabilité du dual est indépendante de x^* , l'inégalité suivante est bien valide : $f(x) + cy \geq f(x) + u(a - g(x))$. Par ailleurs, le théorème de la dualité assure en plus que la valeur optimale u^* maximisant $u(a - g(x^*))$ est aussi l'optimum de PE. À partir d'une affectation particulière de x , la coupe constitue donc une inégalité valable pour tout x et élimine non seulement la solution particulière (*nogood*) qui a permis sa déduction mais aussi toute une classe de solutions impossibles pour les mêmes raisons. Le problème maître s'écrit à la I^{eme} itération :

$$\begin{aligned} \text{PM : Min } & z \\ \text{Tel que : } & z \geq f(x) + u_i^*(a - g(x)) \quad \forall i < I \end{aligned}$$

Il faut donc être en mesure de définir les variables duales (multiplicateurs) au sens de la dualité classique pour pouvoir appliquer la décomposition. Néanmoins, [HO03] propose de pallier cette limitation et d'élargir la notion de dual couramment utilisée en introduisant un *inference dual* pour tout type de sous-problème. Il se place dans un cadre plus général : une décomposition de Benders s'appuyant sur la logique. La dualité se réfère alors à la capacité de produire une preuve, la preuve logique de l'optimalité du sous-problème et de la validité de la coupe qui peut prendre une forme différente de l'inégalité sur x . Cette preuve est obtenue dans le cas classique à travers les théorèmes de la dualité.

Dans le contexte d'un sous-problème discret de satisfaction de contraintes, la résolution du dual consiste à prouver l'infaisabilité du sous-problème et à déterminer les conditions sous-lesquelles cette preuve reste valide (à inférer des coupes). Le succès de l'approche dépend à la fois de la qualité des coupes obtenues mais aussi des structures du problème exploitées par la décomposition (sous-problèmes faciles et indépendants). [HO03] propose d'identifier les classes de problèmes structurés, qui présentent de bonnes caractéristiques. Les problèmes d'ordonnement entrent dans cette catégorie et [JG01] montre ainsi l'efficacité d'une telle décomposition sur un problème de machines parallèles. Un problème linéaire en nombres entiers est considéré comme maître, et la preuve de l'optimalité du dual est réalisée par programmation par contraintes et des techniques de consistance performantes développées pour l'ordonnement. Il obtient ainsi des résultats qui dépassent de loin ceux obtenus avec chacune des techniques indépendamment.

Notre approche se situe exactement dans ce cadre et s’inspire des techniques utilisées pour intégrer la programmation par contraintes dans un schéma de décomposition de type Benders [Tho01, BGR02]. La décomposition du problème sera faite selon ses contraintes (placement et ressources d’une part, temporelles d’autre part) sans partitionnement explicite des variables. Le sous-problème vérifie l’ordonnançabilité d’une allocation, en cas de preuve de son infaisabilité, il identifie *pourquoi* et fournit une information destinée à éliminer toutes les solutions qui partagent ces caractéristiques (un ensemble de contraintes symboliques et arithmétiques). Notre approche rejoint ainsi Benders sur cet élément central : la coupe de Benders. La preuve que nous proposons ici s’appuie sur des techniques d’analyse d’ordonnançabilité de la communauté temps réel. On pourrait penser qu’une preuve analytique (donc rapide) ne puisse fournir une information suffisamment pertinente sur l’inconsistance. Or, c’est de la qualité de cette information dont dépend la vitesse de convergence. À cette fin, nous couplerons ces techniques analytiques avec un algorithme de détection de conflits : QuickXplain [Jun01].

4 Stratégie de résolution

La résolution suppose une coopération étroite entre maître et esclave. Les deux problèmes se basent sur une modélisation commune introduite dans la section suivante pour échanger facilement de l’information. Nous aborderons ensuite le fonctionnement du problème esclave, les mécanismes de coopération à travers l’échange de *nogoods* et de contraintes ainsi qu’une résolution incrémentale exploitant les solutions précédentes.

4.1 Résolution du problème maître

Le problème maître est ici abordé par programmation par contraintes. Le modèle s’appuie une formulation redondante utilisant trois types de variables. Nous considérerons n variables entières x (variables de décision) correspondant à chaque tâche et prenant pour valeur un numéro de processeur : $\forall i \in \{1..n\}, x_i = [1..m]$. Des variables booléennes y indiquent la présence d’une tâche sur un processeur : $\forall i \in \{1..n\}, \forall p \in \{1..m\}, y_{ip} = \{0, 1\}$. Enfin, on introduit également des variables booléennes w explicitant si deux tâches sont situées sur le même processeur ou non : $\forall c_{ij} = (t_i, t_j) \in \mathcal{C}, w_{ij} = \{0, 1\}$.

Le problème maître a aussi pour objectif de répondre efficacement à la problématique de placement qui pose problème à la communauté temps réel. Deux types de contraintes sont à sa charge dans cet objectif : placement et ressources.

- **Résidence (cf. équation (5))** : Il s’agit de valeurs interdites pour x . On peut simplement retirer ces valeurs du domaine initial de x ou poser pour tout processeur interdit p de la tâche t_i : $x_i \neq p$
- **Co-résidence (cf. équation (6))** : $\forall (t_i, t_j) \in \beta^2, x_i = x_j$
- **Exclusion (cf. équation (7))** : $\text{alldifferent}(x_i | t_i \in \gamma)$
- **Capacité mémoire (cf. équation (2))** : $\forall p \in \{1..m\}, \sum_{i \in \{1..n\}} y_{ip} \times m_i \leq \mu_p$
- **Facteur d’utilisation (cf. équation (3))** : on note $\text{ppcm}(T)$ le plus petit commun multiplicateur de l’ensemble des périodes des tâches du système. La contrainte

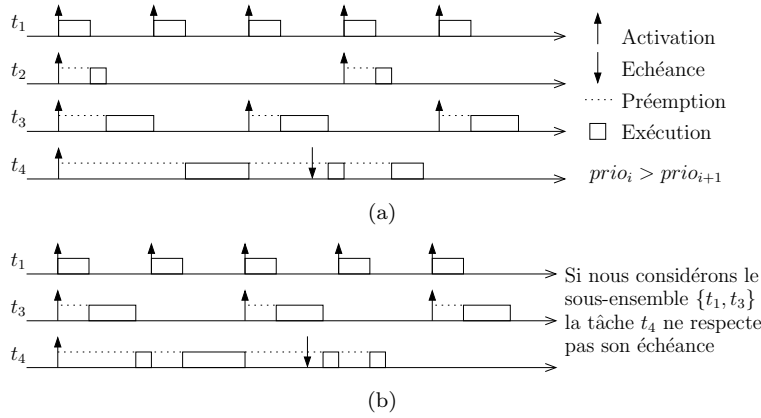


FIG. 2 – Illustration de l'analyse d'ordonnancement. La tâche t_4 ne respecte pas son échéance et le sous ensemble $\{t_1, t_3, t_4\}$ justifie la non-ordonnancement du système.

s'écrit :

$$\forall p \in \{1..m\}, \quad \sum_{i \in \{1..n\}} \frac{ppcm(T) \times WCET_i \times y_{ip}}{T_i} \leq ppcm(T)$$

- **Utilisation du réseau (cf. équation (4)) :** le bus du réseau possède une capacité limitée δ . Dès lors, la taille de l'ensemble des messages échangés ne doit pas dépasser cette limite.

$$\sum_{i \in \{1..n\}} \frac{ppcm(T) \times d_{ij} \times w_{ij}}{T_i} \leq ppcm(T) \times \delta$$

Des contraintes d'intégrité (*channeling constraint*) sont utilisées pour maintenir la consistance des différents modèles. Les liens entre les variables x , y et w sont assurés à l'aide de contraintes de type *element*. Le problème de l'ordonnancement d'un placement est relégué au problème esclave que nous abordons à présent.

4.2 Résolution du problème esclave

L'affectation fournie par le maître constitue une allocation valide, est-elle ordonnable ?

4.2.1 Analyse de l'ordonnancement

Cas des tâches indépendantes. Les premières analyses d'ordonnancement ont été proposées par Liu et Lalaynd [LL73] pour des systèmes temps réel monoprocésseur à priorités fixes avec des tâches indépendantes ($\mathcal{C} = \emptyset$) dont les échéances sont inférieures ou égales aux périodes. L'analyse consiste à calculer pour chaque tâche t_i son pire temps de réponse, $WCRT_i$, en construisant un scénario d'exécution qui pénalise au maximum l'exécution de t_i . Dans le cas des systèmes de tâches indépendantes, il a été prouvé que le pire scénario pour une tâche t_i survient lorsque toutes les tâches plus prioritaires sur

le même processeur se réveille au même instant que t_i (Figure 2). Le pire temps de réponse de t_i est alors :

$$WCRT_i = WCET_i + \sum_{t_j \in hp(A, t_i)} \left\lceil \frac{WCRT_i}{T_j} \right\rceil WCET_j \quad (8)$$

$hp(A, t_i)$ désigne ici l'ensemble des tâches plus prioritaires que t_i sur le processeur $A(t_i)$ pour un placement A donné. Le calcul de $WCRT_i$ s'obtient facilement par la recherche du point fixe de l'équation (8). La comparaison des pires temps de réponse avec les échéances (dans notre cas les périodes) suffit alors pour connaître l'ordonnabilité du système.

Cas des tâches communicantes avec réseau à jeton. Nous devons évaluer dans un premier temps le plus grand délai possible d'émission des données sur le réseau, TRT (*token rotation time*). Cette durée est égale au temps nécessaire pour transmettre l'ensemble des données sur le réseau (au pire une donnée est produite lorsque le processeur sur lequel la tâche émettrice est placée perd le jeton et doit attendre un tour complet) :

$$TRT = \sum_{\substack{\{c_{ij} = (t_i, t_j)\} \\ A(t_i) \neq A(t_j)}} \frac{d_{ij}}{\delta} \quad (9)$$

Une fois TRT évalué nous associons à chaque tâche une échéance en respect avec ce qui a été énoncé en 2.2.1. Une condition nécessaire d'ordonnabilité s'écrit :

$$\forall i = 1..n, WCET_i + \sum_{t_j = hp(A, t_i)} \left\lceil \frac{D_i}{T_j} \right\rceil WCET_j \leq D_i \quad (10)$$

4.3 Coopération entre maître et esclave(s)

Considérons à présent une affectation complète ou partielle des variables x , y , w . Il s'agit de trouver toutes les valeurs de x pour lesquelles la preuve particulière de l'infaisabilité (pour une affectation de x donnée) reste valable, et ceci sans devoir incriminer tout le système. L'information dont on dispose dans tous les cas est l'inconsistance de l'affectation courante, c'est-à-dire un *nogood*. Les relations entre le concept de *nogood* [SV94] propre à la programmation par contraintes et la coupe de Benders propre au milieu de l'optimisation sont mises en évidence dans [HOTK00].

Cas des tâches indépendantes. On peut voir la procédure comme la résolution de m sous-problèmes pour chaque processeur. L'ordonnabilité d'un processeur k est établie en appliquant successivement l'équation (8) à chaque tâche t_i résidant sur k ($x_i = k$) par ordre de priorité décroissante. Dès l'obtention d'une contradiction sur une échéance de la tâche en cours de traitement, on obtient un sous-ensemble de tâches non ordonnables. L'ensemble (t_1, t_2, t_3, t_4) de la figure 2 n'est pas ordonnable, il permet d'expliquer l'inconsistance mais n'est pas minimal. L'ensemble (t_1, t_3, t_4) est suffisant et plus l'explication est précise, plus l'apprentissage est rapide. Dès lors, un algorithme de détection de conflits *QuickXplain* [Jun01] a été utilisé pour minimiser cet ensemble de tâches. Les tâches sont ajoutées à partir de t_1 jusqu'à l'obtention d'une contradiction sur une échéance de t_c , la dernière tâche t_c ajoutée appartient donc au conflit minimal c

(inclus dans (t_1, \dots, t_c)). L'algorithme reprend alors en ajoutant dès le début les tâches incluses dans c . Quand c est inconsistant, il représente un conflit minimal inclus dans l'ensemble conflictuel initial : (t_1, \dots, t_c) .

Le sous-ensemble minimal de tâches incriminées $T \subset \mathcal{T}$ se traduit par une contrainte de type *NotAllEqual*³ sur les variables x :

$$\text{NotAllEqual}(x_i | t_i \in T)$$

On peut noter qu'une telle contrainte pourrait aussi se formuler comme une inégalité sur y pour un processeur donné. Cependant, *NotAllEqual* (x_1, x_2, x_3) élimine les solutions qui rassemblent ces tâches sur le même processeur quel qu'il soit.

Cas des tâches communicantes avec réseau à jeton. Trois types d'informations sont analysées pour proposer une explication d'échec pertinente :

1. On fait dans un premier temps abstraction du réseau. Si un processeur n'est pas ordonnançable sans considérer les temps d'attente supplémentaires dûs aux échanges de messages, il ne le sera pas non plus dans le cas général. On extrait ici à nouveau : *NotAllEqual* $(x_i | t_i \in T)$.
2. On se focalise dans un deuxième temps sur le réseau. Il s'agit de vérifier que toutes les tâches émettrices sur le réseau ont une période inférieure à TRT, sans quoi le jeton ne revient pas assez vite pour assurer leur exécution et l'équation (10) ne sera jamais vérifiée. On identifie ici un ensemble de messages $M \subset \mathcal{C}$ inconsistants :

$$\sum_{c_{ij} \in M} w_{ij} < |M|$$

3. Le dernier test consiste à vérifier l'équation (10). Son échec permet d'obtenir un ensemble $T' \subset \mathcal{T}$ de tâches situées sur des processeurs différents dont l'affectation courante est inconsistante. Cette information est analogue à un *nogood*. Une contrainte particulière a été ici implémentée pour exploiter les symétries du problème et interdire non seulement ce *nogood* mais aussi toutes les permutations des groupes de tâches sur les processeurs. Il s'agit d'une conjonction de *NotAllEqual*.

$$\text{nogood}(x_i | t_i \in T') = \bigwedge_{k \in [1..m]} \text{NotAllEqual}(x_i | t_i \in T' \wedge x_i = k)$$

À l'image du cas des tâches indépendantes, les informations données aux points 1 et 2 sont affinées par QuickXplain. La question qui se pose à présent est de savoir comment intégrer efficacement l'information apprise au fil des échecs ? [Tho01] souligne ce problème et note la possibilité de calculs redondants pour le problème maître. A cette fin, nous traiterons le problème maître comme un problème dynamique.

4.3.1 Résolution incrémentale

Les techniques de résolution dédiées aux problèmes dynamiques se répartissent souvent en deux grandes catégories : les techniques proactives et les techniques réactives.

³*NotAllEqual* porte sur un ensemble V de variables et assure que deux variables au moins parmi V prennent des valeurs distinctes.

Dans un cas, le but est de construire des solutions robustes au changement et dans l'autre, de réutiliser les inférences et les résultats passés pour retrouver une solution de manière incrémentale. Les explications pour la programmation par contraintes s'inscrivent dans cette dernière démarche [?] et offrent une résolution *incrémentale* en conservant la trace de toutes les déductions faites par le solveur au cours de la recherche.

Définition 1 Une explication enregistre toute l'information nécessaire à justifier le comportement du solveur (une réduction de domaine ou une contradiction). Elle est constituée d'un ensemble de contraintes C' (sous-ensemble des contraintes originelles C) et d'un ensemble de décisions dc_1, dc_2, \dots prises durant la recherche. L'explication du retrait de la valeur a de la variable v s'écrit ainsi :

$$C' \wedge dc_1 \wedge dc_2 \wedge \dots \wedge dc_n \Rightarrow v \neq a$$

Une contradiction survient au moment où le domaine d'une variable se vide. L'explication de cette contradiction est calculée par l'union de toutes les explications de chaque valeur de la variable concernée. Dès lors, des algorithmes de backtrack intelligent remettant en question une décision intervenant dans l'explication de la contradiction, sont concevables. En conservant une partie de ces explications d'échecs, on peut mettre en œuvre un mécanisme d'apprentissage.

Les explications interviennent pour la résolution incrémentale de notre problème. À chaque itération entre les problèmes maître et esclave, l'ajout de nouvelles contraintes sont la cause de contradictions. Le processus de backtrack habituel ne se produit pas. À sa place, une *réparation* de la solution est effectuée à partir des décisions mises en cause dans les contradictions à l'image de MAC-DBT [JDB00]. S'il s'avère que des tâches placées au tout début de la recherche précédente ne sont pas ordonnançables, elles seront déplacées sans bouger les tâches placées ultérieurement. Par ailleurs, la phase de re-modélisation prévoit une transformation du modèle et les explications offrent dans ce contexte des facilités de retrait et d'ajout dynamiques de contraintes [?].

Le problème maître n'est résolu qu'une seule et unique fois mais se trouve ainsi continuellement mis à jour et *réparé* par le solveur pour intégrer l'information apprise par le problème esclave.

4.3.2 Renforcement du modèle

La reconnaissance de *pattern* au sein d'un ensemble de contraintes exprimant des problématiques spécifiques est un élément critique de la phase de modélisation. Réaliser cette reconnaissance de pattern de façon automatique entre dans le cadre de l'apprentissage de contraintes [CBQ03]. On aimerait ici exploiter une idée analogue en cherchant à extraire au sein d'un ensemble de contraintes élémentaires, des patterns correspondant à des contraintes globales. Par exemple, l'extraction de *alldifferent* parmi un ensemble de différences est une problématique bien connue et résolue par la recherche d'une clique de taille maximum sur le graphe des contraintes de différences. Nous avons implémenté à cette fin une version de l'algorithme de Bron et Kerbosh [BK73]. De manière similaire, un ensemble de *NotAllEqual* peut se ramener à une contrainte globale de cardinalité : *global cardinality constraint* (*gcc*) [Rég96]. L'obtention du *gcc* pose de nombreuses difficultés et nous n'avons pas encore obtenu de réponse satisfaisante à ce problème mais il constitue une piste clef pour améliorer les résultats, notamment dans des cas d'inconsistance du point de vue de l'ordonnançabilité.

5 Premiers résultats expérimentaux

5.1 Tâches indépendantes

L'avantage de traiter les tâches indépendamment est de pouvoir considérer différents sous-problèmes et incriminer des ensembles de tâches sur chaque processeur. [JG01] souligne la même idée en résolvant un problème de machines parallèles indépendantes.

Pour les problèmes de placement, il n'existe pas de benchmark au sein de la communauté temps réel. En général, les expérimentations sont menées sur des exemples *didactiques* [TBW92, AH98] ou à partir d'un générateur aléatoire de configurations [Ric02, Ram90, MBD98]. Nous avons choisi cette dernière solution. Les paramètres d'entrée du générateur sont :

- n, m : le nombre de tâches et de processeurs (par la suite : $n = 40$ et $m = 7$) ;
- $\%_{global}$: le pourcentage sur le facteur global d'utilisation ;
- $\%_{mem}$: la surcapacité mémoire ;
- $\%_{res}$: le pourcentage de tâches ayant une contrainte de résidence ;
- $\%_{co-res}$: le pourcentage de tâches ayant une contrainte de co-résidence ;
- $\%_{exc}$: le pourcentage de tâches ayant une contrainte d'exclusion ;

Les périodes des tâches ainsi que leur priorité sont obtenues de façon aléatoire. Les temps d'exécution sont tirés aléatoirement et réévalués pour que :

$$\sum_{i=1}^n \frac{WCET_i}{T_i} = m\%_{global}$$

Le besoin mémoire d'une tâche est proportionnel à son pire temps d'exécution. Les capacités mémoires des processeurs sont obtenues aléatoirement et respectent :

$$\sum_{k=1}^m \mu_k = (1 + \%_{mem}) \sum_{i=1}^n m_i$$

Pour pouvoir caractériser nos expérimentations, nous avons défini plusieurs catégories de problèmes dépendant de la dureté respective du placement et de l'ordonnancement. Des notes de 1 à 4 qualifient cette difficulté. La dureté de l'ordonnancement est évaluée sur le facteur global d'utilisation, $\%_{global}$, qui varie de 40 à 90 %. Celle du placement est basée sur le nombre des tâches incluses dans des contraintes de résidence, co-résidence et exclusion ($\%_{res}$, $\%_{co-res}$ et $\%_{exc}$). Par ailleurs, des seuils permettent de jouer sur le nombre de tâches incluses dans les contraintes de résidence et exclusion. La surcapacité mémoire, $\%_{mem}$, allouée à l'ensemble des processeurs entre aussi en jeu (une très faible surcapacité conduit à résoudre un problème de *packing* qui peut devenir très difficile). Le tableau 1 résume ces catégories. Une catégorie notée 1-4 signifie une difficulté 1 pour le placement et 4 pour l'ordonnancement.

La table 2 résume les résultats de nos expérimentations. *NbIter* est le nombre d'itérations entre maître et esclave, *NbNotAlleq* et *NbDiff* sont le nombre de contraintes *NotAllequal* et différences apprises. *CPU* désigne le temps de résolution en secondes. *Xplain* indique si la méthode QuickXplain est utilisée, ou non. Enfin *% Succès* désigne le nombre d'instances résolues avec succès (solution ordonnançable trouvée ou preuve de l'inconsistance effectuée) dans la limite de temps de 10 minutes par instance. Ces données sont une moyenne (sur les instances résolues dans le temps imparti) obtenue

Ordo.	% <i>global</i>		Plac.	% <i>mem</i>	% <i>res</i>	% <i>co-res</i>	% <i>exc</i>
1	40		1	80	0	0	0
2	60		2	40	15	15	15
3	75		3	30	25	25	25
4	90		4	15	35	35	35

TAB. 1 – Définition des catégories de difficulté des problèmes

sur la résolution de 100 instances par catégorie sur un pentium 4,3 GigaHz avec PaLM [JL02] version Java.

Cat(Pl/Or)	Xplain	NbIter	NbNotAllEq	NbDiff	CPU (s)	% Succès
1-1	N	46,35	91,29	4,45	0,58	100%
1-1	O	10,59	39,79	12,41	0,28	100%
1-2	O	26,75	96,93	28,50	3,46	99%
1-3	O	65,23	213,87	39,21	28,70	94%
1-4	O	100,88	373,08	57,82	93,40	40%
2-2	O	46,00	168,27	23,13	34,51	91%
2-3	O	58,89	233,63	37,06	71,18	81%
3-4	O	138,29	131,22	40,65	62,12	91%

TAB. 2 – Résultats obtenus en moyenne sur des instances générées aléatoirement dans des catégories de problèmes (100 problèmes par catégorie)

La catégorie 1-4 représente la catégorie la plus difficile, en l'absence de problème de placement, l'espace initial est complet et tout doit être appris. Par ailleurs, nous sommes ici à la frontière de l'inconsistance du point de vue de l'ordonnabilité. Les limites de la technique sont atteintes dans ce cas sans une reformulation du problème prenant en compte les *NotAllEqual* à travers des contraintes de type *gcc*.

Une instance particulière dans la catégorie 2-3 est détaillée figure 3. On peut observer l'évolution du temps de résolution et du nombre de contraintes apprises à chaque itération. L'apprentissage est très rapide grâce à l'incrémentalité de la résolution qui adapte la solution courante. La solution *pivote* (zone *a-b*) dans un voisinage restreint et le nombre de coupes extraites décroît jusqu'à la formulation d'un problème dur de satisfaction (point *b*) qui oblige le problème maître à changer radicalement de région de l'espace pour fournir une solution faisable. Le processus recommence alors avec un apprentissage à nouveau très rapide (*b-c, c-d, ...*).

5.2 Tâches incluses dans un réseau de communication

Nous avons choisi de tester la technique sur une instance célèbre du milieu de l'ordonnement temps réel, l'instance décrite par Tindell [TBW92] et résolue à l'aide d'un algorithme de recuit simulé. Cette instance présente des particularités notables : le réseau joue en effet un rôle critique dans la mesure où les seules solutions faisables correspondent à une occupation du bus réseau quasi-minimale. Nous sommes parvenus à la résoudre en spécialisant notre approche générique sur ce point particulier : L'optimisation sur l'occupation réseau et l'utilisation d'heuristiques de placement des tâches. La ligne A du

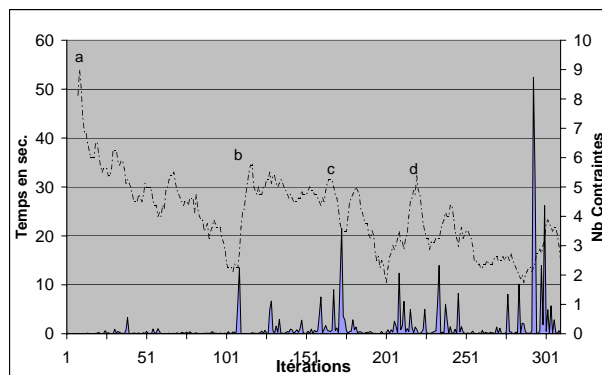


FIG. 3 – Détail d’exécution d’une instance de la catégorie 2-3 comprenant le temps de résolution ainsi qu’une moyenne glissante (pas de 10) du nombre de coupes extraites à chaque itération (en pointillés). (310 itérations, 1192 *NotAllEqual*, 75 *différences* en partie reformulée en 12 *alldifferent*, près de 7 min. de résolution pour obtenir une solution)

tableau 3 correspond à une stratégie de placement qui rassemble les tâches s’échangeant des messages, la ligne B conjugue deux heuristiques dans un schéma *BestFirst* puis *First-Fail* pour obtenir en premier lieu une borne inférieure de qualité sur l’occupation réseau. Les coupes extraites permettent de faire pivoter la solution obtenue en minimisant le réseau vers une solution ordonnançable.

	NbIter	NbNotAllEq	NbNetworkCut	NbNogood	CPU (s)
A	23	26	15	50	314
B	5	4	0	6	8,2

TAB. 3 – Résolution de l’instance de Tindell par deux stratégies *ad hoc*

6 Discussion

La stratégie que nous avons décrite cherche à utiliser la décomposition de Benders comme un moyen de générer intelligemment des *nogoods*. Ce processus s’apparente au schéma d’hybridation *Branch and check* décrit par [Tho01] qui consiste à déléguer la vérification (de la faisabilité) d’une partie du problème à un sous-problème. La principale différence réside dans la décomposition qui est faite selon les contraintes au lieu des variables. Les deux ensembles de contraintes correspondent à des problèmes de différentes natures et le deuxième est utilisé pour générer des explications d’échecs qui s’intègrent dans le premier. L’ordonnançabilité est progressivement convertie en problème d’affectation. L’idée est que le premier problème peut être efficacement pris en compte en programmation par contraintes, en particulier avec un processus de re-modélisation performant. Par ailleurs, on évite le *thrashing* sur les inconsistances dues à l’ordonnançabilité. À l’image des algorithmes basés sur les explications (MAC-DBT ou Decision-Repair [JL02]), une forme d’apprentissage par l’échec s’opère. La technique est actuellement

complète mais il pourrait être intéressant de relâcher cette complétude. Une limite actuelle est la surcharge du mécanisme de propagation avec l'accumulation de contraintes ayant un faible pouvoir de filtrage. Pourquoi ne pas uniquement conserver définitivement en mémoire les *nogoods* qui participent à un modèle plus fort ? On pourrait aussi imaginer construire un algorithme de filtrage s'appuyant sur l'équation (8) ; Cependant, notre objectif est de valider une approche que nous souhaitons mettre en œuvre sur des modèles plus complets d'ordonnancement temps-réel. Les techniques d'analyse dans ce domaine devenant vite très complexes, l'obtention d'une contradiction par une contrainte encapsulant une telle analyse semble moins pertinente qu'une explication de cette contradiction. L'idée est ainsi de tirer profit des connaissances pointues développées en temps réel, dans un cadre de décomposition comme celui de Benders dans lequel la programmation par contraintes permettrait de prendre efficacement en compte le problème de placement.

7 Conclusion et perspectives

Nous avons présenté une approche par décomposition en programmation par contraintes, fondée dans une certaine mesure sur les idées au cœur de la décomposition de Benders et son extension pour tout type de sous-problème. Elle met en œuvre une dualité *logique*, tente de renforcer le modèle en cours de résolution et réalise une résolution incrémentale du problème maître. Dans le cas des tâches indépendantes, l'utilisation d'un algorithme de détection de conflits (*QuickXplain*) est critique pour accélérer la convergence de l'algorithme. Si les limites de l'approche semblent atteintes pour les problèmes inconsistants, nous pensons pouvoir pallier cette difficulté à travers une re-modélisation plus efficace. En s'attaquant à des problèmes de communication entre tâches, on remet en question l'indépendance des sous-problèmes au sein de chaque processeur (un point clef de la décomposition de Benders). Dès lors, pour valider l'approche dans un contexte d'échange de messages, il est crucial d'établir des tests complets sur un panel représentatif d'instances dans ce domaine. Nous projetons également d'étendre notre étude à d'autres types de réseaux (CAN, TDMA, ...) ainsi qu'à la prise en compte des liens de précédences. Enfin, un autre type de contrainte intervient parfois dans le placement : la disjonction entre ensembles de tâches. Cette contrainte globale mal connue semble intéressante pour fournir des outils de résolution et de modélisation performants du problème de placement pour l'ordonnancement temps réel.

Notre approche constitue une réponse nouvelle à une problématique posée au sein de la communauté temps réel et ouvre des perspectives sur l'intégration, dans un cadre Benders, de techniques de résolutions d'horizons plus large que le milieu de l'optimisation.

Références

- [AH98] Peter Altenbernd and Hans Hansson. The Slack Method : A New Method for Static Allocation of Hard Real-Time Tasks. *Real-Time Systems*, 15 :103–130, 1998.
- [Ben62] J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4 :238–252, 1962.
- [BGR02] T. Benoist, É. Gaudin, and B. Rottembourg. Constraint programming contribution to benders decomposition : A case study. In *CP'02*, pages 603–617, 2002.
- [BK73] Coen Bron and Joep Kerbosch. Algorithm 457 : finding all cliques of an undirected graph. *Commun. ACM*, 16(9) :575–577, 1973.

- [BM94] G. Borriello and D. Miles. Task Scheduling for Real-Time Multiprocessor Simulations. *11th Workshop on RTOSS*, pages 70–73, 1994.
- [CBQ03] Rémi Coletta, Christian Bessière, and Joël Quinqueton. Modélisation semi-automatique par acquisition de contraintes. In *9ièmes Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'03)*, pages 129–143, Amiens, France, 2003.
- [FCO99] E. Ferro, R. Cayssials, and J. Orozco. Tuning the Cost Function in a Genetic/Heuristic Approach to the Hard Real-Time Multitask-Multiprocessor Assignment Problem. *Proceeding of the Third World Multiconference on Systemics Cybernetics and Informatics*, pages 143–147, 1999.
- [HKL91] M. González Harbour, M.H. Klein, and J.P. Lehoczky. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. *Proceeding of the IEEE Real-Time Systems Symposium*, pages 116–128, December 1991.
- [HO03] J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96 :33–60, 2003.
- [HOTK00] J.N. Hooker, G. Ottosson, E. S. Thorsteinnsson, and H. Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, special issue on AI/OR*, 15(1) :11–30, 2000.
- [JDB00] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, pages 249–261, Singapore, September 2000. Springer-Verlag.
- [JG01] Vipul Jain and I. E. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13 :258–276, 2001.
- [JL02] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1) :21–45, July 2002.
- [Jun01] 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.
- [Law83] E. L. Lawler. Recent Results in the Theory of Machine Scheduling. *Mathematical Programming : The State of the Art*, pages 202–233, 1983.
- [LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real Time Environment. *Journal ACM*, 20(1) :46–61, 1973.
- [MBD98] Y. Monnier, J.-P. Beauvais, and A.-M. Déplanche. A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System. *24th Euromicro Conference*, 2, 1998.
- [Ram90] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. *10th International Conference on Distributed Computing Systems*, pages 108–115, 1990.
- [Rég96] J.C. Régis. Generalized arc consistency for global cardinality constraint. *AAAI / IAAI*, pages 209–215, 1996.
- [Ric02] Michaël Richard. *Contribution à la Validation des Systèmes Temps Réel Distribués : Ordonnancement à Priorités Fixes & Placement*. PhD thesis, Université de Poitiers, November 2002.
- [San96] F. E. Sandnes. A hybrid genetic algorithm applied to automatic parallel controller code generation. *8th IEEE Euromicro Workshop on Real-Time Systems*, pages 70–75, 1996.
- [SV94] Thomas Schiex and Gérard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *IJAIT*, 3(2) :187–207, 1994.
- [TBW92] 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.
- [TC94] K. Tindell and J. Clark. Holistic scheduling Analysis for Distributed Hard Real-Time Systems. *Euromicro Journal*, pages 40–117, 1994.
- [Tho01] Erlendur S. Thorsteinnsson. Branch-and-check : A hybrid framework integrating mixed integer programming and constraint logic programming. In *CP'01*, 2001.