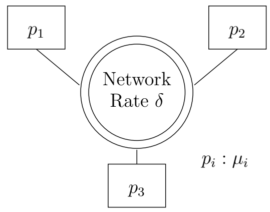
Decomposition and learning for a hard real time task allocation problem

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

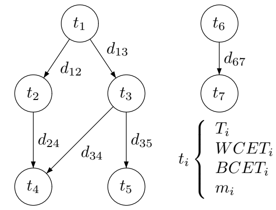
1 Introduction

Real-time systems are at the heart of embedded systems and have applications in many industrial areas : telecommunication, automotive, aircraft and robotics systems, etc. Today, applications (*e.g.* cars) involve many processors to serve different demands (cruise control, ABS, engine management, etc.). These systems are made of specialized and distributed processors (processors are interconnected on a network) which receive data from sensors, process appropriate answers and send it to controllers. Their main characteristics lie in functional as well as non-functional requirements like physical distribution of the resources and timing constraints. Timing constraints are usually specified as deadlines for tasks which have to be executed. Serious damage can occur if deadlines are not met. In this case, the system is called a *hard* real-time system and timing predictability is required. In this field, some related works are based on off-line analysis techniques that compute the response time of the constrained tasks. Such techniques have been initiated by Liu and al. [LL73] and consist in computing the worst-case scenario of execution. Extensions have been introduced later to take into account shared resources, distributed systems [TC94] or precedence constraints [HKL91].

Our problem consists in assigning periodic and preemptive tasks with fixed priorities (a task is periodically activated and can be preempted by a higher priority task) to distributed processors. A solution is an allocation of the tasks on the processors which



(a) Hardware architecture



(b) Software architecture

FIG. 1 – Main parameters of the problem

meets the schedulability requirement. The problem of assigning a set of hard preemptive real-time tasks in a distributed system is NP-Hard [Law83]. It has been tackled with heuristic methods [FCO99, Ram90], simulated annealing [TBW92, BM94] and genetic algorithms [FCO99, San]. However, these techniques are often incomplete and can fail in finding any feasible assignment even after a large computation time. New practical approaches are still needed.

We propose a decomposition based method which separates the allocation problem itself from the scheduling one. It is related to decomposition approaches as the Benders one and especially to the logic Benders based decomposition. On the one hand, constraint programming offers competitive tools to solve the assignment problem, on the other hand, real-time scheduling techniques are able to achieve an accurate analysis of the schedulability. Our method uses Benders decomposition as a way of generating precise nogoods in constraint programming.

This paper is organized as follows : Section 2 introduces the problem.Related work and solving strategies are discussed in Section 3. The logical Benders decomposition scheme is shortly presented and the links with our approach are put forward. Section 4 is dedicated to the master/subproblems and communication between them thanks to nogoods. Experimental results are presented in Section 5 and finally, a discussion of the technique is made in Section 6.

2 Problem description

2.1 Real-time system architecture

The hard real-time system we consider can be modeled with a software architecture (the set of tasks) and a hardware architecture (the physical execution plateform for the tasks). Such a model is used by Tindell [TBW92].

2.1.1 Hardware architecture.

The hardware architecture is made of a set $\mathcal{P} = \{p_1, \ldots, p_k, \ldots, p_m\}$ of m processors with a fixed memory capacity μ_k, connected to a network. All the processors from set $\mathcal{P}$ have the same task processing speed. We consider them identical. Processors are connected to a network with a rate of δ (the amount of data able to transit per unit of time) and a token ring protocol. Such a protocol is usally used by practical networks (due to its deterministic behaviour) : a processor is able to transmit data on the network only if it holds the token. The token travels around the ring from processor to processor and stays at the same place during a fixed maximum period of time. When the time is elapsed or when the processor has finished to send data, the token is sent to the next processor. The maximum period of time ensures the messages waiting on processors to be sent.

2.1.2 Software architecture.

To model the software architecture, we consider a valued, oriented and acyclic graph $(\mathcal{T},\mathcal{C})$. The set of nodes $\mathcal{T} = \{t_1, ..., t_n\}$ corresponds to the tasks whereas the set of edges $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ refers to the communication links between tasks. An edge $c_{ij} = (t_i, t_j) \in \mathcal{C}$ means that a message is sent from t_i to t_j.

A task t_i is defined by its temporal characteristics and resource needs : its period, T_i (a task is periodically activated) ; its worst-case execution time without preemption, $WCET_i$ and its memory need m_i. Edges $c_{ij} = (t_i, t_j) \in \mathcal{C}$ are valued with the amount of exchanged data : d_{ij}. Tasks are able to communicate in two ways : a local communication with no delay using the memory of the processor (requiring the tasks to be located on the same processor) and a distant communication using the network (we assume that two tasks exchanging messages have the same period). In any case, we do not consider precedence constraints. Tasks are periodically activated in an independent way, they read and write data at the beginning and the end of their execution.

Finally, each processor is scheduled with a fixed priority strategy. A level of priority, $prio_i = i$ is given to each task. t_j has priority over t_i if and only if $prio_j < prio_i$. An instance of a task may be pre-empted by a higher priority task and waits for all tasks with higher priority to finish their execution.

2.2 The allocation problem

An allocation is an application $A : \mathcal{T} \to \mathcal{P}$ mapping a task t_i to a processor p_k :

$$t_i \mapsto A(t_i) = p_k \tag{1}$$

The allocation problem consists in finding the application A which respects the constraints described below.

2.2.1 Timing constraints.

They are expressed by the means of deadlines for the tasks. It enforces the duration between the activation date of any instance of the task and its completion time to be bounded by its period T_i. Let TRT (Token Rotation Time) be the maximum duration for sending data on the network for a task t_i, its deadline D_i (the constraint on D_i is detailed in 4.2.1) is then equal to $T_i - TRT$ if t_i sends data to a task allocated on another processor, T_i otherwise.

2.2.2 Resource constraints.

Three kinds of resources are considered :

– **Memory capacity** : The memory usage of a processor p_k cannot not exceed its capacity (μ_k) :

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

- **Utilization factor** : The utilization factor of a processor cannot exceed its processing capacity. The ratio $r_i = WCET_i/T_i$ means that a processor is used r_i% of the time by the task t_i. The following inequality is a simple necessary condition of schedulability :

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

- **Network use** : To avoid overload, the amount of data carried along the network per unit of time cannot exceed the network capacity.

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

2.2.3 Allocation constraints.

Allocation constraints are due to the system architecture and concerns the assignment of tasks to processors. We distinct three kinds of constraints : residence, co-residence and exclusion.

- **Residence** : A task needs sometimes a specific hardware or software resource which is only available on specific processors (*e.g.* a task monitoring a sensor has to run on a processor connected to the input peripheral). It is a couple (t_i, α) where $t_i \in \mathcal{T}$ is a task and $\alpha \subseteq \mathcal{P}$ is the set of available processors for the task. A given allocation A must respect : $A(t_i) \in \alpha$
- **Co-residence** : This constraint enforces several tasks to be placed on the same processor (they share a common resource). Such a constraint is defined by a set of tasks $\beta \subseteq \mathcal{T}$ and any allocation A has to fulfil :

$$\forall (t_i, t_j) \in \beta^2, A(t_i) = A(t_j) \tag{5}$$

- **Exclusion** : Some tasks may be replicated for fault tolerance and therefore cannot be assigned to the same processor. It corresponds to a set $\gamma \subseteq \mathcal{T}$ of tasks which cannot be placed together. An allocation A must satisfy :

$$\forall (t_i, t_j) \in \gamma^2, A(t_i) \neq A(t_j) \tag{6}$$

An allocation is said to be *valid* if it satisfies allocation and resource constraints. It is said to be *schedulable* if it satisfies timing constraints. A solution for this problem is a valid and schedulable allocation of the tasks.

2.3 Off-line and Real-time scheduling

Scheduling problems are often modeled by means of jobs, relations among them, processors and additional resources. The aim is to find an optimal or sub-optimal schedules *w.r.t* a given criterion (makespan, maximum lateness, etc). Actually, a solution consists in assigning to each task, a processor and a start time ; it can therefore be represented with a Gantt chart. The real-time problem differs from the previous one on two points. First, tasks are not statically scheduled and it is impossible to fix in advance the release date of a task. Second, it is not a an optimization problem but a satisfaction one : failures to meet the deadlines are not allowed because they may be catastrophic. Notice that, even if it is called a real-time problem, the allocation is considered as an off-line decision problem.

3 About related decomposition approaches

Our approach is based to a certain extent on a Benders decomposition [Ben62] scheme. We will therefore introduce it to highlight the underlying concepts. Benders decomposition can be seen as a form of *learning from mistakes*. It is a solving strategy that uses a partition of the problem among its variables : x, y. The strategy can be applied to a problem of this general form :

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

A master problem considers only a subset of variables x (often integer variables, D is a discrete domain). A subproblem (SP) tries to complete the assignment on y and produces a Benders cut added to the master problem. This cut has the form $z \geq h(x)$ and constitutes the key point of the method, it is inferred by the dual of the subproblem. Let us consider an assignment x^* given by the master, the subproblem (SP) and its dual (DSP) can be written as follows :

$$\begin{array}{ll} \text{SP : Min } cy & \text{DSP : Max } u(a - g(x^*)) \\ \text{s.t } Ay \geq a - g(x^*) \text{ with : } y \geq 0 & \text{s.t } uA \leq c \text{ with : } u \geq 0 \end{array}$$

Duality theory ensures that $cy \geq u(a-g(x^*))$, $u(a-g(x^*))$ is therefore a lower bound of cy. As feasibility of the dual is independent of x^*, $cy \geq u(a - g(x))$ and the following inequality is valid : $f(x) + cy \geq f(x) + u(a - g(x))$. Moreover, according to duality, the optimal value of u^* maximizing $u(a - g(x^*))$ corresponds to the same optimal value of cy. Even if the cut is derived from a particular x^*, it is valid for all x and excludes a large class of assignments which share common characteristics that make them inconsistent. The number of solutions to explore is reduced and the master problem can be written at the I^{th} iteration :

$$\begin{array}{l} \text{PM : Min } z \\ \text{s.t : } z \geq f(x) + u_i^*(a - g(x)) \qquad \forall i < I \end{array}$$

From all of this, it can be noticed that dual variables or multipliers [1] need to be defined to apply the decomposition. However, [HO03] proposes to overcome this limit and to enlarge the classical notion of *dual* by introducing an *inference dual* available for all kinds of subproblems. He refers to a more general scheme and suggests a different way of thinking about duality : a Benders decomposition based on *logic*. Duality means now to be able to produce a proof, the logical proof of optimality of the subproblem and the correctness of cuts inferred. In original Benders decomposition, this proof is established thanks to duality theorems.

For a discrete satisfaction problem, the resolution of the dual consists in computing the infeasibility proof of the subproblem and determining under what conditions the

[1] Referring to linear programming duality.

proof remains valid. It therefore infers valid cuts.
The success depends on both the degree to which decomposition can exploit structures and the quality of the cuts inferred. [HO03] suggests to identify classes of structured problems that exhibit useful characteristics for the Benders decomposition. Scheduling problems fall into such classes. [JG01] demonstrates that the decomposition approach can be computationally efficient on a scheduling problem with dissimilar parallel machines. An integer linear problem is considered as the master problem. The proofs of optimality of the subproblems are done using strong consistency mechanisms available for scheduling problematics.

Our approach is strongly connected to Benders decomposition and the related concepts. It is inspired from methods used to integrate constraint programming into a benders scheme [Tho01, BGR02]. The problem decomposition will be done among the constraints instead of variables (allocation and ressource from one side, schedulability from the other side). The subproblem checks the schedulability of an allocation, finds out why it is unschedulable and design a set of constraints (symbolical and arithmetic) which rule out all assignments that are unschedulable for the same reason. Our approach concurs therefore the Benders decomposition on this central element : the Benders cut. The proof proposed here is based on off-line analysis techniques from real-time scheduling. One could think that a fast analytic proof could not provide enough relevant informations on the inconsistency. As the speed of convergence and the success of the technique greatly depends on the quality of the cut, a conflict detection algorithm will be coupled with analytic techniques : QuickXplain [Jun01].

4 Solving strategy

The solving process requires a tight cooperation between master and subproblem(s). Both problems share a common model introduced in the next section in order to easily exchange nogoods. Master and subproblem will be presented before examining the cooperation mechanisms made of exchange of nogoods and constraints. An incremental resolution of the master problem which can be seen as a dynamic problem will be finally discussed.

4.1 Master problem

The master problem is solved using constraint programming techniques. The model is based on a redondant formulation using three kinds of variables : x, y, w. At first, let us consider n integer variables x (our decision variables) corresponding to each task and representing the processor selected to process the task : $\forall i \in \{1..n\},\ x_i = [1..m]$. Secondly, boolean variables y indicate the presence of a task onto a processor : $\forall i \in \{1..n\}, \forall p \in \{1..m\},\ y_{ip} = \{0,1\}$. Finally, boolean variables w are introduced to express the fact that a pair of tasks exchanging a message are located on the same processor or not : $\forall c_{ij} = (t_i, t_j) \in \mathcal{C},\ w_{ij} = \{0,1\}$. One of the main objective of the master problem is to efficiently solve the assignment part. It handles two kinds of constraints : allocation and resources.

– **Residence (*cf.* equation (5)) :** it consists in forbidden values for x. A constraint is added for each forbidden processor p of t_i (values can also be retracted from the

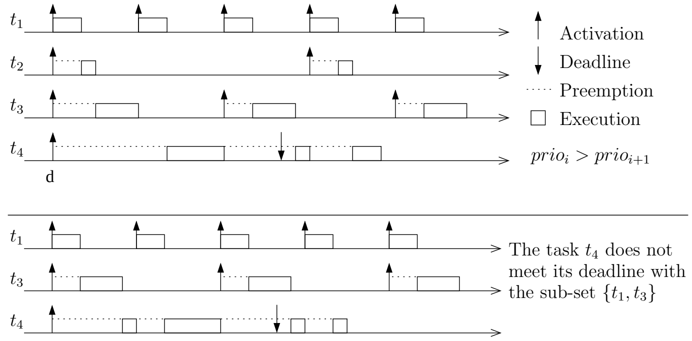


FIG. 2 – Illustration of a schedulability analysis. The task t_4 does not meet its deadline. The sub-set $\{t_1, t_3, t_4\}$ is identified to explain the unschedulability of the system.

initial domain) : $x_i \neq p$

- **Co-residence (*cf.* equation (6)) :** $\forall (t_i, t_j) \in \beta^2, x_i = x_j$
- **Exclusion (*cf.* equation (7)) :** $alldifferent(x_i | t_i \in \gamma)$
- **Memory capacity (*cf.* equation (2)) :** $\forall p \in \{1..m\}$, $\sum_{i \in \{1..n\}} y_{ip} \times m_i \leq \mu_p$
- **Utilization factor (*cf.* equation (3)) :** Let *lcm(T)* be the least common multiple of periods of the tasks. The constraint can be written as follows :

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

- **Network use (*cf.* equation (4)) :** The network capacity is bounded by δ. Therefore, the size of the set of messages carried on the network cannot exceed this limit :

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

Integrity constraints (*channeling constraints*) are added to enforce the consistency of the redondant models. Links between x, y and w are made using *element* constraints.

4.2 Subproblem(s)

An assignment provided by the master problem is a valid allocation of tasks. The problem is here to determine why it may be unschedulable.

4.2.1 Schedulability analysis

Independent tasks. The first schedulability analysis has been initiated by Liu and Layland [LL73] for mono-processor real-time system with independent and fixed priority tasks. The analysis consists in computing for each task t_i its worst response time, $WCRT_i$. The aim is to build the worst execution scenario which penalizes as much as possible the execution of t_i.
For independent tasks, it has been proved that the worst execution scenario for a task

t_i happens when all tasks with a higher priority are awoken simultaneously (date d on Figure 2). The worst-case response time of t_i is :

$$WCRT_i = WCET_i + \sum_{t_j \in hp(A,t_i)} \left\lceil \frac{WCRT_i}{T_j} \right\rceil WCET_j \tag{7}$$

$hp(A, t_i)$ corresponds to the set of tasks with a higher priority than t_i and located on the processor $A(t_i)$ for a given allocation A. $WCRT_i$ is easily obtained by looking for the fix-point of equation (7). Then, it is sufficient to compare the worst case response time with the deadline (in our case, the period of the task) to know if the system is schedulable.

Communicating tasks on a token ring. The result computed by a task must be available to the destination task before the end of the period of the sending task. The messages must reach their destination within the time allowed. It is therefore necessary to bound the maximum delay of transmission on the network (the delay depends on the protocol used). In our case, it corresponds to the token rotation time (TRT). This duration is computed by taking into account all the messages to be sent on the network :

$$TRT = \sum_{\substack{\{c_{ij} = (t_i, t_j) | \\ A(t_i) \neq A(t_j)\}}} \frac{d_{ij}}{\delta} \tag{8}$$

Once TRT has been evaluated, a deadline for each task is computed as described in Section 2.2.1. A sufficient condition of scheduling is written :

$$\forall i = 1..n, WCET_i + \sum_{t_j = hp(A,t_i)} \left\lceil \frac{D_i}{T_j} \right\rceil WCET_j \leqslant D_i \tag{9}$$

4.3 Cooperation between master and subproblem(s)

A complete or partial assignment of variables x, y, w will be now considered. The key point is to find an accurate explanation that encompasses all values of x for which the infeasibility proof (obtained for particular values of x) remains valid. We know at least that the current assignment is contradictory, in other words, a *nogood* is identified. The links between the concept of *nogood* [SV94] introduced in constraint programming and the Benders cut are underlined in [HOTK00].

Independent tasks. m independent subproblems for each processor are solved. The schedulability of a processor k is established by applying equation (8) to each task t_i located on k ($x_i = k$) in a descendent order of priority until a contradiction occurs. For instance, in Figure 2, the set (t_1, t_2, t_3, t_4) is unschedulable. It explains the inconsistency but is not minimal. The set (t_1, t_3, t_4) is sufficient to justify the contradiction and the more precise the deduced explanation is, the more relevant the learning. Therefore, a conflict algorithm, *QuickXplain* [Jun01], has been used to determine the minimal involved set of tasks (w.r.t inclusion). The *propagation* algorithm considered here is equation (7). Tasks are added from t_1 until a contradiction occurs on t_c, the last added task t_c belongs to the minimal conflict c. The algorithm re-starts by initially adding the tasks involved in c.

When c is inconsistent, it represents the minimal conflict among the initial set $(t_1, \ldots, t_c)$. The subset of tasks $T \subset \mathcal{T}$ corresponds to a *NotAllEqual*[2] constraint on x :

$$NotAllEqual(x_i | t_i \in T)$$

It can be noticed that the constraint could be expressed as a linear combination of variables y. However, *NotAllEqual*(x_1,x_3,x_4) excludes the solutions that contain the tasks 1,2,3 gathered on *any* processor.

Communicating tasks on a token ring. The difficulty is to avoid incriminating the whole system :

1. At first, the network is simply not considered. If a processor is unschedulable without taking additionnal latency times due to the exchange of messages, it is still true in the general case. We can again infer : $NotAllEqual(x_i | t_i \in T)$.
2. Secondly, we only consider the network. When the sending tasks have a period less than TRT, the token does not come back early enough to allow the end of their execution. In this case, equation (9) will never be satisfied. A set of inconsistent messages $M \subset \mathcal{C}$ is obtained :

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

3. The last test consists in checking equation (9). A failure returns an inconsistent set $T' \subseteq \mathcal{T}$ of tasks located on different processors. It corresponds to a *nogood*. We use a specific constraint to take advantage of symetries and to forbid this assignment as well as permutations of group of tasks among processors. It can be written as a conjonction of $NotAllEqual$:

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

QuickXplain has been used again to refine informations given in point 2 and 3. Let us now continue with the question of how efficiently integrating the learnt information from the passing failures ? [Tho01] outlines this problem and notices a possible significant overhead with redundant calculations. To address this issue, we considered the master problem as a dynamic problem.

4.3.1 Incremental resolution.

Solving dynamic constraint problem has led to different approaches. Two main classes of methods can be distinguished : proactive and reactive methods. On the one hand, proactive methods propose to build robust solutions that remain solutions even if changes occur. On the other hand, reactive methods try to reuse as much as possible previous reasonings and solutions found in the past. They avoid restarting from scratch and can be seen as a form of learning. One of the main methods currently used to perform such a learning is a justification technique that keeps trace of inferences made by the

[2] A *NotAllEqual* on a set V of variables ensures that at least two variables among V take distinct values.

solver during the search. Such an extension of constraint programming has been recently introduced [Jus03] : explanation-based constraint programming (*e-constraints*).

Definition 1 *An explanation records information to justify a decision of the solver as a reduction of domain or a contradiction. It is made of a set of constraints C' (a subset of the original constraints of the problem) and a set of decisions dc_1,dc_2, ... taken during search. An explanation of the removal of value a from variable v will be written :* $C' \wedge dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \Rightarrow v \neq a$.

When a domain is emptied, a contradiction is identified. An explanation for this contradiction is computed by uniting each explanation of each removal of value of the variable concerned. At this point, intelligent backtracking algorithms that question a relevant decision appearing in the conflict are conceivable. By keeping in memory a relevant part of the explanations involved in conflicts, a learning mechanism can be implemented [JL02].

Here, explanations allow to perform an incremental resolution of the master problem. At each iteration, the constraints added by the subproblem generate a contradiction. Instead of backtracking to the last choice point as usual, the current solution of the master problem is *repaired* by removing the decisions that occur in the contradiction as done by the MAC-DBT algorithm. Tasks assigned at the beginning of the search can be moved without disturbing the whole allocation. In addition, the model reinforcement phase tries to transform a learnt set of elementary constraints with the passing iteration into higher level constraints. Explanations offer facilities to easily dynamically add or remove a constraint from the constraint network [Jus03].

Notice that the master problem is never re-started. It is solved only once but is gradually *repaired* using the dynamic abilities of the explanation-based solver.

4.3.2 Model reinforcement.

Pattern recognition among a set of constraints that expresses specific subproblems is a critical aspect of the modelisation step. Constraint learning deals with the problem of automatically recognizing such patterns. We would like to perform a similar result in order to extract global constraints among a set of elementary constraints. For instance, a set of difference constraints can be formulated as an all-different constraint by looking for a maximal clique in the constraint graph. It is a well-known issue to this problem and a version of the Bron/Kerbosh algorithm [BK73] has been implemented to this end. In a similar way, a set of $NotAllEqual$ constraints can be expressed by a *global cardinality constraint* (gcc) [Rég96]. It is still for us an open question that could significantly improve performances.

5 First experimental results

5.1 Independent tasks

A relevant learning can be made in the case of independent tasks by solving m independent subproblems. [JG01] stresses the same idea by solving an independent parallel machine problem. For the allocation problem, specific benchmarks are not provided in real-time scheduling. Experiments are usually done on didactic examples [TBW92, AH98]

TAB. 1 – Details on classes of difficulty

Ordo.	$\%_{global}$		Alloc.	$\%_{mem}$	$\%_{res}$	$\%_{co-res}$	$\%_{exc}$
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

or random generators of configurations [Ram90, MBD98]. We opted for this last solution. Our generator takes several parameters into account :

- n, m : the number of tasks and processors (experiments have been done on a fixed size : $n = 40$ and $m = 7$) ;
- $\%_{global}$: The global utilization factor of processors.
- $\%_{mem}$: the over-capacity memory, *i.e.* the amount of additionnal memory avalaible on processors with respect to the memory needs of all tasks.
- $\%_{res}$: the percentage of tasks included in residence constraints ;
- $\%_{co-res}$: the percentage of tasks included in co-residence constraints ;
- $\%_{exc}$: the percentage of tasks included in exclusion constraints.

Task periods and priorities are randomly generated. However, worst-case execution time are initially randomly chosen and evaluated again to respect :

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

The memory needs of a task is proportional to its worst-case execution time. Memory capacities are randomly generated but must satisfy :

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

The number of tasks involved in allocation constraints is given by the parameters $\%_{res}$, $\%_{co-res}$, $\%_{exc}$. Tasks are randomly chosen and their number (involved in residence and exclusion constraints) can be set through specific levels. Several classes of problems have been defined depending on the difficulty of both allocation and schedulability problems. The difficulty of schedulability is evaluated using the global usage factor $\%_{global}$ which varies from 40 to 90 %. Allocation difficulty is based on the number of tasks included in residence, co-residence and exclusion constraints ($\%_{res}$, $\%_{co-res}$, $\%_{exc}$). Moreover, the memory over-capacity, $\%_{mem}$ has a significant impact (a very low capacity can lead to solve a *packing* problem, sometimes very difficult). The table 1 describes the parameters and difficulty class of problems. For instance, a class 1-4 indicates an allocation problem of difficulty 1 and a schedulability problem of difficulty 4.

Table 2 summarizes the results of our experiments. *NbIter* is the number of iterations between master and subproblem(s), *NbNotAllEq* and *NbDiff* are the number of *NotAllEqual* and difference constraints inferred. *CPU* is the resolution time in seconds and *Xplain* expresses if the QuickXplain algorithm has been used. Finally % Success gives the number of instances successfully solved (a schedulable solution has been found or the proof of inconsistency has been done) within the time limit of 10 minutes per instance.

TAB. 2 – Average results on 100 instances randomly generated into classes of problems

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

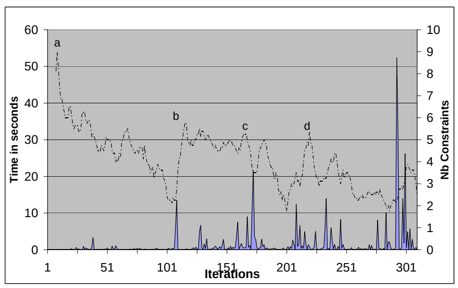


FIG. 3 – Execution of a hard instance of class 2-3. Resolution time and a floating average of step 10 of the number of cuts (in dotlines) inferred at each iteration are shown. (310 iterations, 1192 *NotAllEqual*, 75 *differences* partially re-modeled into 12 *alldifferent* and near 7 minutes of resolution to get a schedulable allocation)

The data are obtained in average (on instances solved within the required time) on 100 instances per class of difficulty with a pentium 4, 3 GigaHz and the java version of PaLM [Jus03].

The class 1-4 represents the hardest class of problem. Without the allocation problem, the initial search space is complete and everything has to be learnt. Moreover, these problems are close to inconsistency due to the hardness of the schedulability. Limits of our approach seem to be reached in such a case without an efficient re-modeling of *NotAllEquals* constraints into *gcc* (see 4.3.2).

The execution of a particular and hard instance of class 2-3 is outlined on Figure 3. Resolution time and learnt constraints at each iteration are detailed. One can notice the speed of the learning process at the beginning due to the incremental resolution which adapts the current solution to the cuts. The solution *moves* in a close neighborhood and the number of cuts decreases until a hard satisfaction problem is formulated (*a*-*b* in Fig. 3). The master problem is then forced to go out of the current region of the search space to provide a valid allocation (*b*). The process starts again with a quick learning of nogoods (*b*-*c*, *c*-*d*).

5.2 Communicating tasks on a token ring.

We choose to experiment the technique on a well-known instance of real-time scheduling : the Tindell instance, solved [TBW92] thanks to simulated annealing. This instance exhibits a particular structure : the network plays a critical part and feasible solutions have a network utilization almost minimal. We were forced to specialize our generic approach on this particular point through the use of allocation heuristics and optimization on the network. The line A of table 3 corresponds to an heuristic that tries to gather tasks exchanging messages. The line B combines two heuristics in a *BestFirst* and *FirstFail* scheme to obtain at first an accurate lower bound on network utilization. Inferred cuts allow to adapt the solution, while minimizing the network, towards a feasible solution. Completeness is therefore lost in this approach.

TAB. 3 – Resolution of the Tindell instance with two *ad hoc* strategies

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

6 Discussion

Our approach tries to use logic based Benders as a mean of generating relevant nogoods. It is not far from the hybrid framework *Branch and Check* of [Tho01] which consists in checking the feasibility of a delayed part of the problem into a subproblem. The main difference lies in the partition of constraints instead of variables. The two sets of constraints refer to problems of different nature and the second set is used to produce precise explanations of failures that make sense for the first one. The schedulability problem is gradually converted into the assignment problematic. The idea is that the first problem could be efficiently taken into account with constraint programming, and especially, with an efficient re-modeling process. In addition, it avoids thrashing on schedulability inconsistencies. As for explanation based algorithms (MAC-DBT or Decision-repair [JL02]), it tries to learn from its mistakes. The technique is actually complete but it could be interesting to relax its completeness (from this point, we step back from Benders). One current problem is the overload of the propagation mechanism because of the accumulation of low power filtering constraints. Why not keeping definitively in memory only nogoods which take part to a stronger model ? (a fine management of memory can be implemented due to dynamic abilities of the master problem). One could also think building a filtering algorithm on equation (7). However, the objective is to show how precise nogoods could be used and to validate an approach we intend to implement on complex scheduling models. As analysis techniques quickly become very complex, a contradiction raised by a constraint encapsulating such an analysis seems to be less relevant than a precise explanation of failure. The idea is to take advantage of the know-how of real-time scheduling community in a decomposition scheme as the Benders one where constraint programming could efficiently solve the allocation problem.

7 Conclusion and future work

We propose in this paper, a decomposition method built to a certain extent on a logic Benders decomposition as a way of generating nogoods. It implements a *logical* duality to

infer nogoods, tries to enforce the constraint model and finally performs an incremental resolution of the master problem. It is also strongly related to a class of algorithms which intends to learn from mistakes in a systematic way by managing nogoods. For independent tasks, the use of QuickXplain is critical to speed up the convergence but the limits seem to be reached for highly constrained and inconsistent problems. Nevertheless, we believe that the difficulty can be overcome through an efficient re-modeling process. Independent subproblems (a key point of Benders) are lost when solving communicating task problems. Experiments on a representative panel of instances have therefore to be carried out. We also plan to extend our study to other kinds of network protocols (CAN, TDMA, etc.) and precedence constraints. From all of this, we can conclude that first experiments have encouraged us to go a step further. It is in particular critical to compare it with other methods as a traditional constraint and linear programming. Finally, another kind of constraints sometimes occur : disjunction between set of tasks. The disjunction global constraint has not been studied a lot and it could provide accurate modeling and solving tools to tackle the assignment problem with more complex allocation constraints.
Our approach gives a new answer to the problematic of real-time task allocation. It opens new perspectives on integrating techniques coming from a broader horizon than optimization, within CP in a Benders scheme.

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, E. 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.

[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 Systelms 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. Thorsteinsson, 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.

[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.

[Jus03] Narendra Jussien. The versatility of using explanations within constraint programming. RR 03-04-INFO, École des Mines de Nantes, France, 2003.

[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. Régin. Generalized arc consistency for global cardinality constraint. *AAAI / IAAI*, pages 209–215, 1996.

[San] F. E. Sandnes. A hybrid genetic algorithm applied to automatic parallel controller code generation. *8th IEEE Euromicro Workshop on Real-Time Systems*.

[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. Thorsteinsson. Branch-and-check : A hybrid framework integrating mixed integer programming and constraint logic programming. In *CP'01*, 2001.