Solving a real-time allocation problem with constraint programming

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

Solving a Real-Time Allocation Problem with Constraint Programming

Pierre-Emmanuel Hladik [a,b,*], Hadrien Cambazard [b], Anne-Marie Déplanche [a], Narendra Jussien [b]

[a] *Université de Nantes, IRCCyN, UMR CNRS 659, 1 rue de la Noë – BP 9210, 44321 Nantes Cedex 3, France*

[b] *École des Mines de Nantes, LINA CNRS, 4 rue Alfred Kastler – BP 20722, 44307 Nantes Cedex 3, France*

Abstract

In this paper, we present an original approach (CPRTA for "Constraint Programming for solving Real-Time Allocation") based on constraint programming to solve a static allocation problem of hard real-time tasks. This problem consists in assigning periodic tasks to distributed processors in the context of fixed priority preemptive scheduling. CPRTA is built on dynamic constraint programming together with a learning method to find a feasible processor allocation under constraints. Two efficient new approaches are proposed and validated with experimental results. Moreover, CPRTA exhibits very interesting properties. It is complete (if a problem has no solution, the algorithm is able to prove it); it is non-parametric (it does not require specific tuning) thus allowing a large diversity of models to be easily considered. Finally, thanks to its capacity to explain failures, it offers attractive perspectives for guiding the architectural design process.

Key words: hard real-time task allocation, fixed priority scheduling, constraint programming, global constraint, Benders decomposition

1 Introduction

Real-time systems have applications in many industrial areas: telecommunication systems, the automotive and aircraft industries, robotics, etc. Today's applications are becoming more and more complex, not only in their software (an

* Corresponding author. Tel: 00 33(0)251-8582-25 Fax: 00 33(0)251-8582-49
Email address: Pierre-Emmanuel.Hladik@emn.fr (Pierre-Emmanuel Hladik).

increasing number of concurrent tasks with various interaction schemes) and their execution platform (many distributed processing units interconnected through specialized network(s)), but also in their numerous functional and non-functional requirements (timing, resources, power, etc.). One of the main issues in the architectural design of such complex distributed applications is to define an allocation of tasks onto processors so as to meet all the specified requirements. Even if it has to be solved off-line most of the time, it needs efficient and adaptable search techniques, which can be integrated into a more global design process. Furthermore, it is desirable that these techniques return relevant information intended to help the designer who is faced with architectural choices. In particular the "binary" result (has a feasible allocation been found? Yes and here it is, or no, and that is all) which is usually returned by the search algorithm is not satisfactory in failure situations. The designer expects some explanations justifying the failure and enabling him to revisit his design. Therefore, more sophisticated search techniques that can collect some knowledge about the problem they solve are required. These are the general objectives of the work we are conducting.

More precisely, the problem we are concerned with consists in assigning a set of periodic, pre-emptive tasks to distributed processors in the context of fixed priority scheduling not only to respect schedulability but also to account for requirements related to memory capacity, co-residence, redundancy, and so on. The mapping we are concerned with is static, *i.e.* all allocation decisions are made off-line, before the application runs, and they do not change afterwards. We assume that the characteristics of tasks (execution time, priority, etc.) and those of the physical architecture (processors and network) are all known *a priori*.

In the literature, the allocation problem is studied extensively. A classification of related works is difficult because of the varied nature of the hardware architecture model (multiprocessor, distributed, homogeneous, etc.), the software architecture model (periodic tasks, deadline, precedence relations, etc.), the set of constraints (memory, allocation constraints, etc.), the objective (to minimize execution and communication costs, to balance the load, etc.), and the solving strategy.

Since the problem of allocating tasks is generally NP-hard [34], some form of enumerative method or approximation using heuristics needs to be developed for this problem: graph theory techniques [57,38,18], branch-and-bound [41,45,44,50,65], genetic algorithm [40,6,53,2,39,19,20,42,21], clustering [4,1,36], steepest descent (or Hill climbing) [40,64], tabu search [46,64]; simulated annealing [63,11,40,14,15,17,19,64], neural network [56,2], and dedicated heuristics [54,17,32,66,3]. However, today, for a specific problem, no technique seems to be more appropriate than another.

Moreover, the majority of these techniques suffer from two main drawbacks: firstly, their implementation is widely conducted by the considered models or objective, so it is difficult to add a new constraint or to consider a new model; secondly, their performances are sensitive to initial parameters and need experimental tuning.

That is why we decided to tackle the allocation problem with a quite innovative approach, constraint programming (cp). The main advantages of cp are: its *declarativity*, the variables, domains, constraints are simply described; its *genericity*, it is not a problem-dependent technique, general rules are mechanically performed during the search; its *adaptability*, each constraint can be considered as independent and a model could be simply extended by merging these different constraints; and its *non-parametric* ability. This technique has been widely used to solve a large range of combinatorial problems. It has proved quite effective in a variety of applications (from planning and scheduling to finance – portfolio optimization – through biology). Up to now, solving the allocation problem of hard real-time tasks with cp has not been studied extensively. To our knowledge, only Szymanek *et al.* [58], Schild and Würtz [55] and Ekelin [16] have used cp to produce an assignment and a pre-runtime scheduling of distributed systems under optimization criteria. Even though their context is different from ours, their results have shown the ability of such an approach to solve an allocation problem for embedded systems and have encouraged us to go further.

In this paper, two approaches are considered. The first one introduces a global constraint [1] and an *ad hoc* algorithm, *i.e.* a filtering algorithm, to tackle schedulability. This algorithm is a custom-written filtering algorithm designed to take into account and to exploit the structure of the schedulability constraints. The second investigated approach uses the complementary strengths of constraint programming and optimization methods from operational research like numerous hybridation schemes [61,28,26,8]. It is a decomposition-based method (related to logic-based Benders decomposition [27]), which separates the allocation problem from the scheduling one: the allocation problem is solved by means of *dynamic constraint programming* tools, whereas the scheduling problem is treated with specific real-time schedulability analysis. The main idea is to "learn" from the schedulability analysis to re-model the allocation problem so as to reduce the search space. In that sense, we can compare this approach to a form of *learning from failures*.

First experimental results show that these two methods produce an efficient way to solve the allocation problem. Moreover, a fundamental property of these methods is their completeness: when a problem has no solution, they are able to prove it (contrary to heuristic methods that are unable to decide).

[1] A constraint is said to be global when it is a conjunction of a set of constraints.

Moreover, they are also able to produce an explanation when they fail to find a solution. As will be shown in the paper, this explanation could be used in the future as a way to help the designer of real-time systems during the design process.

The remainder of this paper is organized as follows. In Section 2, we describe the problem. Some related works are presented in Section 3. Section 4 briefly introduces constraint programming before translating the problem as a constraint satisfaction one in Section 5. The two approaches we propose are then described: Section 6 is concerned with the global constraint and Section 7 is dedicated to the logical Benders decomposition. Some experimental results are presented in Section 8. Section 9 shows how it is possible to set up a failure analysis able to aid the designer to review his plans. It is a first attempt that proves its feasibility but it will need to go deeper. The paper ends with concluding remarks in Section 10.

2 Description of the problem

We base our research on the model in [63] with few differences as the communication protocol and the schedulability test.

2.1 The real-time system architecture

The hard real-time system we consider can be modeled by a software architecture, the set of tasks, and a hardware architecture, the execution platform for the tasks, as represented in Fig. 1.

By *hardware architecture*, we mean a set $\mathcal{P} = \{p_1, \ldots, p_k, \ldots, p_m\}$ of m processors with fixed memory capacity m_k and identical processing speed. Each processor schedules tasks assigned to it with a fixed priority strategy. It is a simple rule: a static priority is given to each task and, at run-time, the ready task with the highest priority is put in the running state, pre-empting a lower priority task. These processors are fully connected through a communication medium with a bandwidth δ. In this paper, we look at a communication medium called a *CAN bus*, which is currently used in a wide spectrum of real-time embedded systems. However, any other communication network could be considered if its timing behavior (including its protocol rules) is predictable. Thus, the first experiments we conducted addressed a token ring network [24].

CAN (Controller Area Network) [10] is both a protocol and a physical network.

CAN works as a broadcast bus, meaning that all connected nodes can read all messages sent on the bus. Each message has a unique identifier, which is also used as the message priority. On each node, waiting messages are queued. The bus makes sure that when a new message is selected to transfer, the message with the highest priority, waiting on any connected node, will be transmitted first. When at least one bit of a message has started to be transferred it cannot be pre-empted even though higher priority messages arrive. As a result, the CAN behavior will be seen subsequently as that of a non-pre-emptive fixed priority message scheduling.

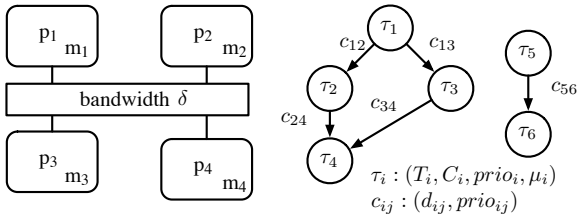

Fig. 1. An example of hardware (left) and software (right) architecture.

The *software architecture* is modeled as a valued, oriented and acyclic graph $(\mathcal{T}, \mathcal{C})$. The set of nodes $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ represents the tasks. A task in turn is a set of instructions that must be executed sequentially in the same processor. The set of edges $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ refers to the data sent between tasks.

A task τ_i is defined through timing characteristics and resource needs: its period T_i (as a task is periodically activated, the date of its first activation is free), its worst-case execution time without pre-emption C_i and its memory need μ_i. A priority $prio_i$ is given to each task. Task τ_j has priority over τ_i if, and only if, $prio_i < prio_j$. In our approach, we treat allocation as an issue separate from that of scheduling. Priority assignment is globally performed before allocation. Rate monotonic or deadline monotonic algorithms can be used. However, our approach does not consider a specific scheduling policy.

Edges $c_{ij} = (\tau_i, \tau_j) \in \mathcal{C}$ are weighted with the amount of exchanged data d_{ij} together with a priority value $prio_{ij}$ (useful in the CAN context) [2]. In this model, we assume that communicating tasks have the same activation period but we do not consider any precedence constraint between them. They are periodically activated in an independent way, and they read input data and write output data at the beginning and the end of their execution.

The underlying communication model is inspired by OSEK-COM specifications [43]. OSEK-COM is a uniform communication environment for automo-

[2] Task priorities are all assumed to be different. The same assumption is made about message priorities.

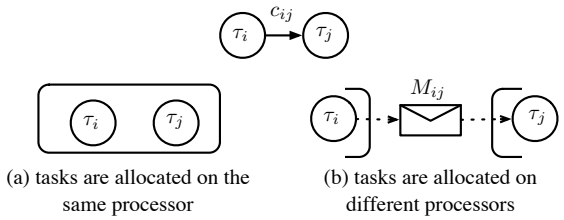

Fig. 2. Depending on the task allocation, a message exists or not.

tive control unit application software. It defines common software communication interface and behavior for internal communications (within an electronic control unit) and external ones (between networked vehicle nodes) which are independent of the communication protocol used. It is the following. Tasks that are located on the same processor communicate through local memory sharing. Such a local communication cost is assumed to be zero. On the other hand, when two communicating tasks are assigned to two distinct processors, the data exchange needs the transmission of a message on the network. Here we are interested in the *periodic transmission mode* of OSEK-COM. In this mode, data production and message transmission are not synchronized: a producer task writes its output data into a local unqueued buffer from where a periodic protocol service reads it and sends it into a message. The building of protocol data units considered here is very simple: each piece of data d_{ij} that has to be sent from a producer task τ_i to a consumer task τ_j in a distant way gives rise to its own message M_{ij}. Moreover, in this paper, for the sake of simplicity, the *asynchronous receiving mode* is preferred. It means that the release of a consumer task τ_j is strictly periodic and unrelated to the M_{ij} message arrival. Thus, when a node receives a message from the bus, its protocol records its data into a local unqueued buffer from where it can be read by the task τ_j. In [25], an extension of this work to a *synchronous receiving mode* is proposed in which a message reception notification activates the consumer task.

As a result, depending on the task allocation, an edge c_{ij} of the software architecture may give rise to two different equivalent schemes as illustrated in Fig. 2. In Fig. 2(b), M_{ij} inherits its period T_i from τ_i and its priority $prio_{ij}$ from c_{ij}.

Therefore, from a scheduling point of view, messages on the bus are very similar to tasks on a processor. Like for tasks, each message M_{ij} is "activated" every T_i units of time; its (bus) priority is $prio_{ij}$; and it has a transmission time C_{ij} (the time it takes to transfer the message on the bus, see Section 2.2.3 for its computation).

2.2 The allocation problem

An allocation is a mapping $A : \mathcal{T} \rightarrow \mathcal{P}$ such that:

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

The allocation problem consists in finding a mapping A that respects the whole set of constraints described below.

2.2.1 Resource constraints

Three kinds of constraints are considered [3] :

- **Memory capacity**: The memory use of a processor p_k cannot exceed its capacity (m_k):
$$\forall k = 1..m, \sum_{A(\tau_i)=p_k} \mu_i \leq m_k \tag{2}$$
- **Utilization factor**: The utilization factor of a processor cannot exceed its processing capacity. The following inequality is a necessary schedulability condition:
$$\forall k = 1..m, \sum_{A(\tau_i)=p_k} \frac{C_i}{T_i} \leq 1 \tag{3}$$
- **Network use**: To avoid overload, the messages carried along the network per unit of time cannot exceed the network capacity:
$$\sum_{\substack{c_{ij} = (\tau_i, \tau_j) \\ A(\tau_i) \neq A(\tau_j)}} \frac{s_{ij}}{T_i} \leq \delta \tag{4}$$
where s_{ij} is the message size; it is a function of d_{ij} depending on the message structure (see Section 2.2.3).

2.2.2 Allocation constraints

Allocation constraints are due to the system architecture. We distinguish three kinds of constraint.

- **Residence**: a task may need a specific hardware or software resource that is only available on specific processors (*e.g.* a task monitoring a sensor has to run on a processor connected to the input peripheral). This constraint is

[3] Precise units are not specified but obviously they have to be consistent with the given expressions.

expressed as a couple (τ_i, α) where $\tau_i \in \mathcal{T}$ is a task and $\alpha \subseteq \mathcal{P}$ is the set of available host processors for the task. A given allocation A must respect:

$$A(\tau_i) \in \alpha \tag{5}$$

- **Co-residence**: several tasks may have to be assigned to 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:

$$\forall(\tau_i, \tau_j) \in \beta^2, A(\tau_i) = A(\tau_j) \tag{6}$$

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

$$\forall(\tau_i, \tau_j) \in \gamma^2, A(\tau_i) \neq A(\tau_j) \tag{7}$$

2.2.3 Timing constraints

Timing constraints are expressed by means of the relative deadlines for the tasks and messages. A timing constraint enforces the duration between the activation date of any instance of the task τ_i and its completion time to be bounded by its relative deadline D_i. Depending on the task allocation, such timing constraints may concern the instantiated messages too. For tasks as well as messages, their relative deadline is hereafter assumed equal to their activation period.

A widely chosen approach for the schedulability analysis of a task and message set S is based on the following necessary and sufficient condition [35]: S is schedulable if, and only if, for each task and message of S, its worst-case response time is less than or equal to its relative deadline. Thus, it leads us to compute worst-case response times for the tasks on the processors and for the messages on the bus. According to the features of the considered task and message models, as well as the processor and bus scheduling algorithms, a classical computation can be used and its main results are given below.

Task worst-case response time. For independent and periodic tasks with a pre-emptive fixed priority scheduling algorithm, it has been proved that the worst-case execution scenario for a task τ_i happens when it is released simultaneously with all the tasks that have a priority higher than $prio_i$. When D_i is (less or) equal to T_i, the worst-case response time for τ_i is given by [35]:

$$R_i = C_i + \sum_{\tau_j \in hp_i(A)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{8}$$

where $\lceil x \rceil$ is the smallest integer greater than x and $hp_i(A)$ is the set of tasks with a priority higher than $prio_i$ and located on the processor $A(\tau_i)$ for a given allocation A. The summation gives the time that tasks with higher priority will take before τ_i has completed. The worst-case response time R_i can be easily solved by looking for the fix-point of Eq. (8) in an iterative way.

Message worst-case response time. As mentioned earlier, message scheduling on the CAN bus can be viewed as a non-pre-emptive fixed priority scheduling strategy. Thus, when writing the worst-case response time equation for a message, Eq. (8) has to be reused with some modifications. First, it has to be changed so that a message can only be pre-empted during its first transmitted bit instead of its whole execution time. Second, a blocking time, *i.e.* the longest time the message might be blocked by a lower priority message, must be added. The resulting worst-case response time equation for the CAN message M_{ij} is [62]:

$$R_{ij} = C_{ij} + L_{ij} \tag{9}$$

with

$$L_{ij} = \sum_{M' \in hp_{ij}(A)} \left\lceil \frac{L_{ij} + \tau_{bit}}{T'} \right\rceil C' + \max_{M' \in lp_{ij}(A)} \{C' - \tau_{bit}\} \tag{10}$$

where $hp_{ij}(A)$ (respectively $lp_{ij}(A)$) is the set of messages derived from the allocation A with a priority higher (respectively lower) than $prio_{ij}$; τ_{bit} is the transmission time for one bit (τ_{bit} is related to the bus bandwidth δ, τ_{bit} (second) $= 1/\delta$ (bit per second)); C' is the worst-case transmission time for the message M'.

Here as well the computation of Eq. (10) can be solved iteratively.

To calculate the worst-case transmission time C_{ij} of a message, Eq. (11) is used [62]:

$$C_{ij} = s_{ij}\tau_{bit} \tag{11}$$

with

$$s_{ij} = \left\lfloor \frac{34 + 8d_{ij}}{5} \right\rfloor + 47 + 8d_{ij} \tag{12}$$

It shows that the message size is not given directly from the data size d_{ij} (in bytes) and the frame overhead of 47 bits (identifier, CRC, etc.). It has to take into account the possible overhead caused by the bit stuffing process of CAN controllers.

From now on, an allocation A is said to be *valid* if it meets allocation and resource constraints. It is *schedulable* if it meets timing constraints. Finally, a solution to our problem is a valid and schedulable allocation of the tasks.

Notice that the objective of this article is to validate an application (find a solution) and not optimize a function such as workload balancing [60,2], the

number processors used [42] nor the response time of tasks [45,2].

3 Related work

In a general way, the task mapping problem includes assigning (allocation) each task to a processor and ordering (scheduling) the execution of the tasks on each one such that all functional and temporal constraints of the system are respected. In the literature, mapping problems are studied extensively and an exhaustive overview, or even a classification of them, is difficult because of the huge variety of problems and solving techniques they consider.

For models quite similar to the one presented in Section 2, [63,11,41,54,4,50,21] are good examples of the research efforts. Like us, they all consider a *fixed priority scheduler*. Allocation and scheduling are usually considered as two independent stages. For most of them [63,11,41,54,4,21], only the allocation problem is solved since task priorities are known. In [50], Richard *et al.* propose a method that simultaneously allocates tasks to processors and assigns priorities to tasks and messages.

Different techniques are used: a simulated annealing algorithm in [63,11], a branch-and-bound search algorithm in [41,50], a genetic algorithm in [21], and a dedicated heuristic in [54]. In [4], a clustering algorithm is proposed. In that case, the solving process is divided into two steps. Firstly, the search space is reduced by merging some tasks according to some specific criteria (this is the clustering step). Secondly, a classic solving algorithm (simulated annealing, branch-and-bound, etc.) is used for allocating task clusters to processors. However, by reducing the search space, clustering suffers from the risk of not finding any feasible assignment.

All these techniques seem effective ways for solving allocation problems. However, they all suffer from two important drawbacks: 1) they are sensitive to their initial parameters, *i.e.* temperature and cooling for simulated annealing, selection criteria and population size for genetic algorithm, enumeration order for branch-and-bound, etc. 2) much work is necessary to efficiently extend a method to a new model, *i.e.* the objective function of the simulated annealing must be redefined, branch-and-bound functions have to be designed for each mapping problem, crossing-over or mutation operators of genetic algorithm are dedicated to only one problem, etc.

Now it seemed to us that a constraint programming (cp) approach should be able to limit, indeed to avoid, these disadvantages. Moreover, it constitutes a new solving strategy for the real-time task allocation problem. It is true that little attention has been paid to it [55,58,59,16]. Furthermore, these dif-

fer fundamentally in the scheduling technique: only cyclic task schedulers are assumed, *i.e.* a scheduling cycle is derived before runtime, and the runtime system dispatches tasks based on current time in respect with the cycle. The fact that only a cyclic scheduling is considered makes the problem of mapping very different from ours. Indeed, the scheduling cycle is viewed as a set of discrete intervals, the constraints of which can be naturally expressed in the classical formalism of cp. The main problem with dynamic scheduling is that it is not possible to simply express the schedulability test as an efficient constraint for a cp solver.

In [55], precedence constraints between tasks as well as end-to-end deadlines are taken into account for computing the scheduling cycle. In [58], more constraints are considered: i) resources (memory and shared resource), ii) communications (a cyclic scheduling is also produced for messages), iii) power consumption. In [16], Ekelin summarizes of all these works and proposes a general approach for multi-constraints and multi-objective functions for real-time cyclic scheduling. As in [4], Szymanek and Kuchcinski [59] studied the problem of task clustering under multi-resource constraints.

The first objective of our work has been to study the applicability of cp to our allocation problem. It has led us to propose an efficient way to include the dynamic scheduling model in the cp solver. From this point of view, this paper makes an innovative contribution in comparison with previous works.

4 A short introduction to constraint programming

A *constraint satisfaction problem* (csp) consists of a set V of variables x_i defined by a corresponding set D_i of possible values (the so-called *domain*) and a set C of constraints. A solution to the problem is an assignment of a value in D_i to each variable x_i in V such that all constraints are satisfied. For example, consider a 3-uple of variables $V = \{x_1, x_2, x_3\}$, their domains $D_1 = D_2 = D_3 = \{1, 2, 3\}$ and two constraints C_1: $x_1 > x_2$ and C_2: $x_1 = x_3$. A solution of this csp is $x_1 = 2$, $x_2 = 1$ and $x_3 = 2$.

The solutions to a csp can be found by systematically searching through the possible assignments of values to variables. The variables are sequentially labeled and, as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If any of the constraints is violated, backtracking is performed to the most recently instantiated variable that still has available values.

However, cp offers more accurate methods to solve a csp. One of them is based on removing inconsistent values from domains of variables till a solution is

found. Consider the previous example: the value 1 could be removed from D_1, because C_1 cannot be respected if $x_1 = 1$. Several consistency techniques exist and they are combined to solve a csp. For example, when a value is removed from the domains of variables, it could be propagated through other constraints. In the previous example, after removing 1 from D_1, 1 could be removed from D_3 because of C_2.

This mechanism coupled with a backtracking scheme allows the search space to be explored in a *complete way*. For a more detailed introduction to cp, we refer the reader to [52].

In this paper, the search is performed by the *Maintaining Arc-Consistency* algorithm (mac). mac is nowadays considered as one of the best algorithms for solving a csp. Moreover, a specific mac algorithm has been used, based on the use of explanations. In a few words, an explanation can be considered as a limited trace of the past activity of the constraint solver. It is a set of constraints, and records enough information to justify any decision of the solver such as a reduction of domain or a contradiction. Due to space limitation, we cannot expand upon this point but the reader may refer to [30] for a detailed presentation.

5 Translation of the allocation problem into a csp

The first thing one has to do when using cp to solve a problem is to word it as a csp. In our case, it relies on a redundant formulation using three sets of variables: x, y, w.

Let us first consider n integer-valued variables x_i which are decision variables, each one corresponding to one task, and representing the processor selected to process the task: $\forall i \in \{1..n\},\ x_i \in \{1, \ldots, m\}$. Then, boolean variables y_{ip} indicate the presence of a task on a processor: $\forall i \in \{1..n\}, \forall p \in \{1..m\},\ y_{ip} \in \{0, 1\}$. Finally, boolean variables w_{ij} are introduced to express whether a pair of tasks exchanging data are located on the same processor or not: $\forall c_{ij} = (\tau_i, \tau_j) \in \mathcal{C},\ w_{ij} \in \{0, 1\}$. Integrity constraints are used to enforce the consistency of the redundant model. This redundant model has been chosen to speed up the search and propagate algorithms.

Moreover, the constraints of our allocation problem have to be mapped on this model. It appears that allocation and resource constraints can be directly expressed with the classical constraints of cp. Their translation into the csp are given here:

- **Resource constraints**

· **Memory capacity:** (*cf.* Eq. (2)) $\forall p \in \{1..m\},\ \sum_{i \in \{1..n\}} y_{ip}\mu_i \leq \mu_p$
· **Utilization factor:** (*cf.* Eq. (3)) Let $\text{lcm}(T)$ be the least common multiple of periods of the tasks [4]. The constraint can be written as follows:

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

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

$$\sum_{i \in \{1..n\} j \in \{1..n\}} \frac{w_{ij}\,\text{lcm}(T)s_{ij}}{T_i} \leq \text{lcm}(T)\delta$$

- **Allocation constraints**
  · **Residence:** (*cf.* Eq. (5)) it consists of forbidden values for x. A constraint is added for each forbidden processor p of τ_i: $x_i \neq p$
  · **Co-residence:** (*cf.* Eq. (6)) $\forall (\tau_i, \tau_j) \in \beta^2, x_i = x_j$
  · **Exclusion** [5]**:** (*cf.* Eq. (7)) *AllDifferent*$(x_i | \tau_i \in \gamma)$

However, because of the schedulability analysis they require, timing constraints cannot be translated as directly as the previous ones. Two approaches have been studied to consider timing constraints. For the first approach, timing constraints are taken into account at each variable assignment the solver makes. Partial schedulability analyses must be conducted and their results transformed into new constraints on variables. This method is introduced in Section 6. Inversely, the second approach consists in breaking down the allocation problem into two subproblems (one that deals with allocation and resource constraints, the other with timing constraints), and managing cooperation between them so as to find a valid and schedulable solution. Section 7 is dedicated to this method.

6 A global constraint for schedulability

To tackle schedulability during search, a classical approach is to integrate it as a constraint of the cp solver. For this purpose, some notations used previously for a full allocation are extended to a partial one:

[4] Utilization factor and network use are reformulated with the lcm of task periods because our constraint solver cannot currently handle constraints with both real coefficients and integer variables.
[5] An *AllDifferent* constraint on a set V of variables ensures that all variables among V have different values.

Definition 6.1 *A partial allocation is a mapping* $a : \mathcal{U} \subset \mathcal{T} \rightarrow \mathcal{P}$ *such that:*

$$\tau_i \in \mathcal{U} \mapsto a(\tau_i) = p_k \tag{13}$$

Definition 6.2 *We call a decision from a partial allocation* $a : \mathcal{U} \rightarrow \mathcal{P}$*, a partial allocation* δ *of a task set* $\Delta \subsetneq \mathcal{U}$*. We denote the new allocation produced as* $a' = a + \delta$*. It is such that* $a' : \mathcal{U} \bigcup \Delta \rightarrow \mathcal{P}$ *and* $\forall \tau_i \in \mathcal{U}, a'(\tau_i) = a(\tau_i)$*,* $\forall \tau_i \in \Delta, a'(\tau_i) = \delta(\tau_i)$

We denote $R_i(a)$ the worst-case response time of τ_i for a partial allocation a. It is the worst-case response time that τ_i would exhibit if only those tasks allocated in a were considered. $hp_i(a)$, respectively $lp_i(a)$ is the set of higher, respectively lower, priority tasks than τ_i on the same processor as $a(\tau_i)$.

6.1 Filtering algorithm

The cp solver proceeds step by step. At each step, a decision is taken from a partial allocation and a filtering algorithm is used. A filtering algorithm associated with a constraint C is an algorithm that may remove some values that are inconsistent with C, but that does not remove any consistent values [49]. For example, consider the same csp as in Section 4: a 3-uple of variables $\{x_1, x_2, x_3\}$, their domains $D_1 = D_2 = D_3 = \{1, 2, 3\}$ and two constraints $C_1 : x_1 < x_2$ and $C_2 : x_1 = x_3$. In the first step, x_1 takes the value 1. The filtering algorithm of C_1 removes the value 1 of D_2 and the filtering algorithm of C_2 removes the values 2 and 3 of D_3. At the next step, x_2 takes the value 2 and so on.

The holy grail with a global constraint is to achieve generalized arc consistency (gac) which simply means a complete filtering algorithm so that any value that does not belong to a solution is eliminated. This involves providing *polynomial* necessary and sufficient conditions regarding the existence of a solution for the constraint (according to current domains of variables).

τ_i	τ_1	τ_2	τ_3	τ_4	τ_5
T_i	20	10	15	15	4
C_i	12	2	5	7	2
$prio_i$	1	2	3	4	5

Table 1
Task and message characteristics

For example, consider a simple example of five tasks τ_i with domain $\{p_1, p_2\}$ such that every triple (10 triples exist) is unschedulable (Table 1 gives an example). Assigning five tasks to two processors will force placing three of them together, which will raise a contradiction as any triple is forbidden. A polynomial filtering algorithm for our global schedulability constraint that

respects the gac should detect this inconsistency. However, we can prove it is impossible to do.

Property 6.1 *Achieving generalized arc-consistency for the schedulability constraint is NP-hard.*

Proof. In [37], Leung and Whitehead proved that (Theorem 2.6, page 242) for a given software architecture $\mathcal{T}$ and a hardware architecture $\mathcal{P}$, the problem of deciding whether an allocation from $\mathcal{T}$ to $\mathcal{P}$ is schedulable is NP-hard. The proof is done by reducing the 3-PARTITION problem to the schedulability problem. The 3-PARTITION is a classical problem that has been shown to be NP-complete and so achieving generalized arc-consistency for the schedulability problem is NP-hard. Moreover, as the schedulability test (Equation 8) is pseudo-polynomial, we cannot conclude whether the problem of gac is NP-complete. □

Our filtering algorithm is therefore based on a relaxation of the gac. Algorithm 1 shows the pseudo-code of the filtering algorithm for the global constraint to tackle schedulability. It begins to check the schedulability of the current partial allocation (line 1). It is easily done by computing the worst-case response time of only allocated tasks and present messages (property 11.1, Appendix 1). A present message is a message that we are sure exists for a partial allocation: consider a communication c_{ij}, the message M_{ij} exists if, and only if, the domains of values of τ_i and τ_j are disjoint ($D_i \cap D_j = \emptyset$). Notice that a message can exist even if its producer or its consumer is not allocated. If the allocation is schedulable, the filtering algorithm removes values that are inconsistent (lines 2 to 13) by testing the schedulability for each value of each remaining variable (line 7). Then, the pruning is propagated within the constraint set until a fix-point is reached (while loop).

This filtering can, however, miss powerful deductions of the gac. With the example presented in Table 1, an inconsistent state will not be detected by our pragmatic approach and a search tree will be built over three tasks among the five to prove this inconsistency, see Figure 3: (1) τ_1 is allocated to p_1. All other tasks could still be allocated to p_1 (loop from line 4 to line 12). (2) τ_2 is also allocated to p_1. The inconsistency is detected because D_3, D_4 and D_5 are reduced to p_2 (line 8) and τ_3, τ_4 and τ_5 are not schedulable on p_2. (3) τ_2 is allocated to p_2. All domains of unallocated tasks are still $\{p_1, p_2\}$. (4) τ_3 is allocated to p_1. The inconsistency is detected because D_4 and D_5 are reduced to p_2 and τ_2, τ_4 and τ_5 are not schedulable on p_2. (5) τ_3 is allocated to p_2. The inconsistency is detected because D_4 and D_5 are reduced to p_1 and τ_1, τ_4 and τ_5 are not schedulable on p_1. (...) and so on. However, this approach is already very costly and we will now discuss ways to speed it up.

Algorithm 1 Filtering algorithm for the schedulability global constraint after a decision δ from a partial allocation a

GLOBAL CONSTRAINT($a' := a + \delta$)

```
1: flag := CHECKSCHEDULABILITY(a') {flag becomes true if a' is schedulable}
2: while flag do
3:    flag := false
4:    for each unallocated task τ_i do
5:       for each p in τ_i's domain do
6:          a'' := a' + δ_1 with δ_1(τ_i) := p
7:          if not CHECKSCHEDULABILITY(a'') then
8:             Remove p from domain of τ_i {Propagate to other constraints}
9:             flag := true
10:         end if
11:      end for
12:   end for
13: end while
```

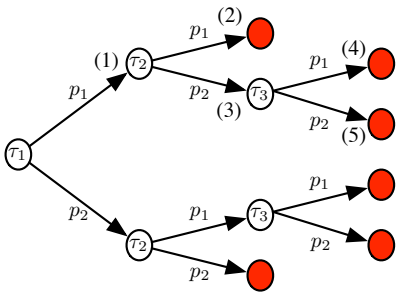

Fig. 3. Tree-search for an allocation with the global constraint. Colored nodes are those where the problem is inconsistent.

6.2 Incrementality

To speed up the filtering of the global constraint for a partial allocation, it is possible to use some knowledge from the previous partial allocations. This incrementality can be used twice in the filtering algorithm. Firstly, during the schedulability test and, secondly, during the propagation by reducing the number of values to check in the domains of variables. These two points are now developed.

6.2.1 Incrementality of the schedulability test

From now on, we suppose that all other constraints are met, and so the partial allocation is valid too. Consider a partial but schedulable allocation a and a

decision from a that allocates only one task τ_i with $a'(\tau_i) = p_k$. Since the scheduling of processors is made in an independent way, only those tasks on p_k may suffer from the allocation of τ_i. Moreover, all tasks in $hp_i(a')$ are still schedulable because the worst-case response time equation (Eq. 8) ensures that all tasks $\tau_j \in hp_i(a')$ keep the same response time: $R_j(a') = R_j(a)$.

Rule 1 *When a task is allocated to a processor from a schedulable partial allocation, schedulability has to be checked only for this task and the lower priority ones on that processor.*

This rule could be extended to a decision δ with $\#\Delta > 1$, where $\#X$ stands for the cardinality of the set X. For the same reason, schedulability has to be checked only for the tasks $\tau_i \in \Delta$, $a'(\tau_i) = p_k$, and the tasks $\tau_j \in \mathcal{U}$ such that $a'(\tau_j) = p_k \wedge prio_i > prio_j$.

The same reasoning can be applied to message schedulability analysis.

Rule 2 *When a new message appears M_{ab}, message schedulability has to be checked only for that message, the lower priority ones and the higher priority ones M_{ij} such that $C_{ab} > \max_{M' \in lp_{ij}(a)}\{C'\}$.*

Rule 2 can be extended to a decision that gives rise to many messages. Let Δ^* be the set of these new messages. After a decision δ, message schedulability has to be checked only for M_{ab}, the highest priority message of Δ^*, the messages in $lp_{ab}(a')$ and the higher priority messages $M_{ij} \in hp_{ab}(a)$ such that $\max_{M \in \Delta^*} C > \max_{M' \in lp_{ij}(a)}\{C'\}$.

These rules can be implemented into the checkSchedulability function called at lines 1 and 7 in algorithm 1.

6.2.2 Incrementality of domains

By considering the previous partial allocation and the current decision, the number of values in the domain of a variable that have to be checked in order to remove inconsistency values (line 5 in algorithm 1) can be reduced.

Let a be a partial allocation and δ a decision such that $a' = a + \delta$, $\Delta^* = \emptyset$ and $\tau_i \notin \mathcal{U} \bigcup \Delta$. If p_k is a processor not used in δ, *i.e.* no task is assigned to it in δ, and τ_i is schedulable on p_k for a, then τ_i is still schedulable on p_k for a'.

If $\Delta^* \neq \emptyset$, because of messages, we cannot conclude about the schedulability of the system without checking the network.

Rule 3 *After a decision δ where $\Delta^* = \emptyset$, the domains of non-allocated tasks to check during filtering are reduced to the processor set where a task has been*

added in δ.

6.3 Reducing further schedulability tests

Other rules can be introduced to reduce domains during filtering by considering some dominance relationships between tasks. A rule can be deduced from property 11.2 (see Appendix 1):

Rule 4 *If, from a schedulable partial allocation, allocating τ_i to p_k makes τ_i unschedulable, then p_k has to be removed from the domain of all tasks τ_j such that $prio_j < prio_i \wedge C_j \geq C_i \wedge T_j \leq T_i$.*

In the same way, the following rule can be stated from property 11.3 (see Appendix 1):

Rule 5 *If, from a schedulable partial allocation, allocating τ_i to p_k makes an allocated task τ_b unschedulable, then p_k has to be removed from the domain of all tasks τ_j with $prio_j > prio_b \wedge C_j \geq C_i \wedge T_j \leq T_i$.*

Finally, a trivial propagation rule can be added by considering message schedulability:

Rule 6 *If, from a schedulable partial allocation a, allocating a communicating task τ_i makes a message unschedulable, then the domain of τ_i has to become $\bigcup_{\tau_j}\{a(\tau_j)\}$ where τ_j is an allocated task that exchanges data with τ_i.*

These rules can be easily implemented in Algorithm 1 on line 8.

7 Solving the problem with logic-based Benders decomposition

Contrary to the global constraint strategy, which includes schedulability in the search algorithm, the approach presented in this section is based on the Benders decomposition and separates resource and allocation constraints from schedulability ones.

7.1 Benders decomposition scheme

We only give the basic principles of this technique, for a more detailed description please refer to [12]. Our approach is based on an extension of a Benders scheme. A Benders decomposition [7] is a solving strategy of linear problems

that uses a partition of the problem among its variables: x, y. A master problem considers only x, whereas a subproblem tries to complete the assignment on y and produces a Benders cut added to the master. This cut is the central element of the technique, it is usually a linear constraint on x inferred by the dual of the subproblem. Benders decomposition can therefore be seen as a form of *learning from mistakes*.

For a discrete satisfaction problem, the resolution of the dual consists in computing the unfeasibility proof of the subproblem (in this case, the dual is called an *inference dual*) and determining under what conditions the proof remains valid to infer valid cuts. The Benders cut can be seen in this context as an explanation of failure that is learnt by the master. We refer here to a more general Benders scheme called *logic Benders decomposition* [27] where any kind of subproblem can be used as long as the inference dual of the subproblem can be solved.

We propose an approach inspired by methods used to integrate constraint programming into a logic-based Benders decomposition [61,8]. The allocation and resource constraints are considered on one side, and timing ones through schedulability on the other (see Fig. 4). The master problem solved with cp yields a valid allocation. The subproblem checks the schedulability of this allocation, eventually finds out why it is unschedulable and designs a set of constraints, named *nogoods*, which rules out all the assignments that are unschedulable for the same reason.

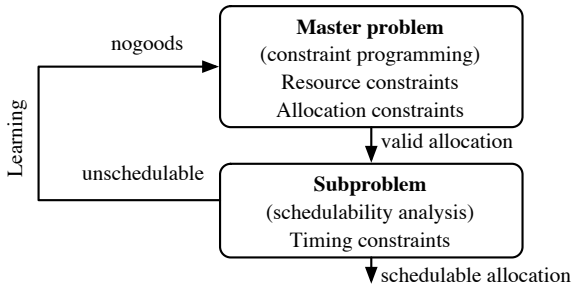

Fig. 4. Logic-based Benders decomposition to solve an allocation problem

7.2 Cooperation between master and subproblem

The subproblem considered here is to check whether a valid solution produced by the master problem is schedulable or not. Schedulability analysis is used (see section 2.2.3). We now consider a valid allocation (like the one the cp solver may propose) in which some tasks are not schedulable. Our purpose is to explain why this allocation is unschedulable, and to translate this into a new constraint for the master problem.

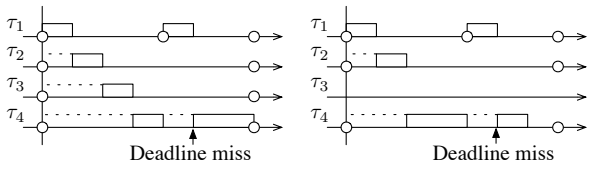

Fig. 5. Illustration of a schedulability analysis. The task τ_4 does not meet its deadline. The subset $\{\tau_1, \tau_2, \tau_4\}$ is identified to explain the unschedulability of the system.

Tasks. The explanation for the unschedulability of a task τ_i is the presence of tasks with higher priority on the same processor that interfere with τ_i. For any other allocation with τ_i and $hp_i(A)$ on the same processor, it is certain that τ_i will still be detected unschedulable. So, the master problem must be constrained so that all solutions where τ_i and $hp_i(A)$ are together are not considered any further. This constraint corresponds to a *NotAllEqual* [6] on x:

$$NotAllEqual(x_j | \tau_j \in S_i(A) = hp_i(A) \cup \{\tau_i\})$$

It is worth noticing that this 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 τ_1, τ_3, τ_4 gathered on *any* processor.

Nevertheless it is easy to see that this constraint is not totally relevant. For example, in Fig. 5, τ_4 that shares a processor with τ_1,τ_2 and τ_3 misses its deadline. Actually, the set $S_4(A) = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ explains the unschedulability but it is not minimal in the sense that, if we remove one task from it, the set is still unschedulable. Here, the set $S_4(A)' = \{\tau_1, \tau_2, \tau_4\}$ is sufficient to justify the unschedulability. To explain the unschedulability of a task, there could be more than one minimal task set, this is dependent on the order of enumeration of $hp_i(A)$. In the example, $\{\tau_1, \tau_3, \tau_4\}$ is also a minimal set. However, the more constraints there are, the slower the solver is, thus we consider just one minimal set in explanation.

In order to derive more precise explanations (to achieve a more relevant learning), a conflict detection algorithm, namely *QuickXplain* [29] (see algorithm 2), has been used to determine a minimal (*w.r.t.* inclusion) set of involved tasks $S_i(A)'$. A new function is defined, $R_i(X)$, as the worst-case response time of τ_i as if it was scheduled with those tasks belonging to the set X that have priority over it:

$$R_i(X) = C_i + \sum_{\tau_j \in hp_i(A) \cap X} \left\lceil \frac{R_i(X)}{T_j} \right\rceil C_j \tag{14}$$

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

Algorithm 2 Minimal task set

QUICKXPLAINTASK(τ_i, A, D_i)

1: $X := \emptyset$
2: $\sigma_1, ..., \sigma_{\#hp_i(A)}$ {an enumeration of $hp_i(A)$. The enumeration order of $hp_i(A)$ may have an effect on the content of the returned minimal task set}
3: **while** $R_i(X) \leq D_i$ **do**
4: $k := 0$
5: $Y := X$
6: **while** $R_i(Y) \leq D_i$ and $k < \#hp_i(A)$ **do**
7: $k := k + 1$
8: $Y := Y \cup \{\sigma_k\}$ {according to the enumeration order}
9: **end while**
10: $X := X \cup \{\sigma_k\}$
11: **end while**
12: **return** $X \cup \{\tau_i\}$

Messages. The reasoning is quite similar. If a message M_{ij} is found unschedulable, it is because of the messages in $hp_{ij}(A)$ and the longest message in $lp_{ij}(A)$. We denote $M_{ij}(A)$ their union together with $\{M_{ij}\}$. The translation of this information in terms of constraint yields:

$$\sum_{M_{ab} \in M_{ij}(A)} w_{ab} < \#M_{ij}(A)$$

It is equivalent to a *NotAllEqual* constraint on a set of messages since, to be met, it requires that at least one message of $M_{ij}(A)$ "disappears" ($w_{ab} = 0$).

Like for tasks, so as to reduce the set of involved messages, QUICKXPLAIN has been implemented, using a similar adaptation of Eqs. (9) and (10). It returns a minimal set of messages $M_{ij}(A)'$.

7.3 Applying the method to an example

An example to illustrate the theory is developed hereafter. It shows how the cooperation between master and subproblems is performed. Table 2 shows the characteristics of the considered hardware architecture (with 4 processors) and Table 3 those of the software architecture (with 20 tasks). The entry "$x, y \rightarrow j$" for the task τ_i indicates an edge c_{ij} with $C_{ij} = x$ and $prio_{ij} = y$.

The problem is constrained by:

- residence constraints:

p_i	p_0	p_1	p_2	p_3
m_i	102001	280295	360241	41617

Table 2
Processor characteristics

τ_i	T_i	C_i	μ_i	$prio_i$	Message
τ_0	36000	2190	21243	1	600,1 → 13
τ_1	2000	563	5855	6	500,3 → 8
τ_2	3000	207	2152	15	600,7 → 7
τ_3	8000	2187	21213	3	
τ_4	72000	17690	168055	7	300,4 → 9
τ_5	4000	667	6670	8	800,5 → 19
τ_6	12000	3662	36253	14	
τ_7	3000	269	2743	16	
τ_8	2000	231	2263	12	100,6 → 18
τ_9	72000	6161	59761	9	
τ_{10}	12000	846	8206	4	200,2 → 15
τ_{11}	36000	5836	60694	20	
τ_{12}	9000	2103	20399	10	
τ_{13}	36000	5535	54243	13	
τ_{14}	18000	3905	41002	18	
τ_{15}	12000	1412	14402	5	
τ_{16}	6000	1416	14301	17	700,8 → 17
τ_{17}	6000	752	7369	19	
τ_{18}	2000	538	5487	11	
τ_{19}	4000	1281	12425	2	

Table 3
Task and message characteristics

  - CC_1: τ_0 must be allocated to p_0 or p_1 or p_2.
  - CC_2: τ_{16} must be allocated to p_1 or p_2.
  - CC_3: τ_{17} must be allocated to p_0 or p_3.
- co-residence constraints:
  - CC_4: τ_7, τ_{17} and τ_{19} must be on the same processor.
- exclusion constraints:
  - CC_5: τ_3, τ_{11} and τ_{12} must be on different processors.

To start the resolution process, the solver for the master problem finds a valid solution in accordance with CC_1, CC_2, CC_3, CC_4 and CC_5. How the cp solver finds such a solution is not our objective here. The valid solution it returns is:

- processor p_0: τ_2, τ_5, τ_7, τ_8, τ_9, τ_{17}, τ_{19}.
- processor p_1: τ_4, τ_6, τ_{12}, τ_{13}.
- processor p_2: τ_0, τ_{11}, τ_{14}, τ_{15}, τ_{16}.
- processor p_3: τ_1, τ_3, τ_{10}, τ_{18}.

One deduces that messages are $M_{0,13}$, $M_{1,8}$, $M_{4,9}$, $M_{8,18}$, $M_{10,15}$, and $M_{16,17}$.

It is easy to check that it is a valid solution by considering allocation and resource constraints:

- $\mu_2 + \mu_5 + \mu_7 + \mu_8 + \mu_9 + \mu_{17} + \mu_{19} = 93383 \leq m_0$;
- $\mu_4 + \mu_6 + \mu_{12} + \mu_{13} = 278950 \leq m_1$;
- $\mu_0 + \mu_{11} + \mu_{14} + \mu_{15} + \mu_{16} = 151642 \leq m_2$;
- $\mu_1 + \mu_3 + \mu_{10} + \mu_{18} = 40761 \leq m_3$;
- $\frac{C_2}{T_2} + \frac{C_5}{T_5} + \frac{C_7}{T_7} + \frac{C_8}{T_8} + \frac{C_9}{T_9} + \frac{C_{17}}{T_{17}} + \frac{C_{19}}{T_{19}} = 0.972 \leq 1$;
- $\frac{C_4}{T_4} + \frac{C_6}{T_6} + \frac{C_{12}}{T_{12}} + \frac{C_{13}}{T_{13}} = 0.938 \leq 1$;
- $\frac{C_0}{T_0} + \frac{C_{11}}{T_{11}} + \frac{C_{14}}{T_{14}} + \frac{C_{15}}{T_{15}} + \frac{C_{16}}{T_{16}} = 0.794 \leq 1$;
- $\frac{C_1}{T_1} + \frac{C_3}{T_3} + \frac{C_{10}}{T_{10}} + \frac{C_{18}}{T_{18}} = 0.894 \leq 1$.
- $\frac{C_{0,13}}{T_0} + \frac{C_{1,8}}{T_1} + \frac{C_{4,9}}{T_4} + \frac{C_{8,18}}{T_8} + \frac{C_{10,15}}{T_{10}} + \frac{C_{16,17}}{T_{16}} = 0.454 \leq 1$.

The subproblem now checks the schedulability of the valid solution. The schedulability analysis proceeds in three steps.

First step: analyzing the schedulability of tasks. The worst-case response time for each task is obtained by application of Eq. (8) and it is compared with its relative deadline. Here τ_5, τ_{12}, τ_{16} and τ_{19} are found unschedulable.

Second step: analyzing the schedulability of messages. The worst-case response time for each message is obtained by application of Eq. (9) and Eq. (10) and it is compared with its relative deadline. Here $M_{1,8}$ is found unschedulable.

Third step: explaining why this allocation is not schedulable. The unschedulability of τ_5 is due to the interference of higher priority tasks on the same processor: $hp_5 = \{\tau_2, \tau_7, \tau_8, \tau_9, \tau_{17}\}$. By applying QuickXPlainTask (see algorithm 2) with hp_5 ordered by increasing index, we find $S_5(A)' = \{\tau_5, \tau_9\}$ as the minimal set. Consequently, the explanation of the unschedulability is translated into the new constraint:

$$CC_6\text{: } \mathit{NotAllEqual}\{x_5, x_9\}$$

In the same way, by applying QuickXPlainTask for τ_{12}, we find:

$$CC_7\text{: } \mathit{NotAllEqual}\{x_6, x_{12}, x_{13}\}$$

for τ_{16}:

$$CC_8\text{: } \mathit{NotAllEqual}\{x_{11}, x_{16}\}$$

and for τ_{19}:

$$CC_9\text{: } \mathit{NotAllEqual}\{x_9, x_{19}\}$$

For $M_{1,8}$, we have:

$$M_{1,8}(A) = \{M_{0,13}, M_{1,8}, M_{4,9}, M_{8,18}, M_{16,17}\}.$$

QUICKXPLAIN returns $M_{1,8}(A)' = \{M_{0,13}, M_{1,8}, M_{4,9}, M_{16,17}\}$ as the minimal set. So another constraint is created:

$$CC_{10}\text{: } w_{0,13} + w_{1,8} + w_{4,9} + w_{16,17} < 4$$

These new constraints CC_6, CC_7, CC_8, CC_9 and CC_{10} are added to the master problem. They define a new problem for which it has to search for a valid solution and so on.

After 20 iterations between the master problem and the subproblem, this allocation problem is proven without solution. This results from 78 constraints learnt all along the solving process. This example has been solved using ŒDIPE (see Section 8). On a computer with a G4 processor (800MHz), its computing time was 10.3 seconds.

8 Experimental results

We have developed a dedicated tool named ŒDIPE [13] that implements our solving approaches. ŒDIPE is based on the CHOCO [33,31] cp system and PALM [30], an explanation-based cp system.

The time that cp takes to find a solution depends on three parameters: the size of the search space, the ability to reduce that search space, and the way it is searched. In cp the search of solutions is managed by a search strategy that aims at quickly directing the search towards good solutions without loosing the completeness. A search strategy is related to the order of choice of variable and value. It consists in defining an algorithm that specifies at each branching the next variable to be chosen, together with its value. Different approaches have been studied in [23] (conduct by the domain sizes, by the task worst case execution times, etc.) and the most efficient strategy we found is a general search strategy for constraint programming inspired by integer programming techniques and based on the concept of impact of a variable [48]. The impact measures the importance of a variable for the reduction of the search space. Impacts are learnt from the observation of domain reduction during search and need no tuning. Experiments have been conduct with this search strategy.

For the allocation problem, no specific benchmarks are available as a reference in the real-time community. Experiments are usually done on didactic

examples [63,4] or randomly generated configurations [47,39]. We adopted the latter solution. Our generator takes several parameters into account:

- **Tasks.** The number of tasks, n, is fixed at 40. Periods are generated to have a limitation of the hyper-period while various values are possible [22]. Priorities are randomly assigned. Worst-case response times are generated with the routine UUniFast [9], which implements an algorithm for efficiently generating unbiased vectors of utilization. A parameter, $\%_{global}$, is used to scale the global utilization such that $\sum_{i=1}^{n} C_i/T_i = m\%_{global}$. The difficulty of a problem related to schedulability is evaluated using $\%_{global}$, which varies from 40 to 90%.
- **Communications.** The number of communications, mes, *i.e.* the number of edges in the task graph, and the network workload, $\%_{mes}$, impact on the difficulty of a problem. For the sake of simplicity, only linear data communications between tasks are considered and the priority of a message is inherited from the task producing it. Worst-case transmission times are generated with the same technique as worst-case execution times such that $\sum_{i=1}^{n} C_{ij}/T_i = \%_{mes}$.
- **Processors.** The number of processors, m, is fixed at 7. The memory capacity of a processor is generated such that $\sum_{k=1}^{m} m_k = (1 + \%_{mem}) \sum_{i=1}^{n} \mu_i$ where $\%_{mem}$ represents the global memory over-capacity for the hardware architecture. The memory over-capacity has a significant impact on the difficulty of a problem (a very low capacity can lead to solving a *packing* problem, sometimes very difficult).
- **Allocation constraints.** The percentage of tasks involved in allocation constraints is given by the parameters $\%_{res}$ for residence constraints, $\%_{co}$ for co-residence, and $\%_{exc}$ for exclusion constraints. Different constraints are sized such that $\%_{res}$, $\%_{co}$ and $\%_{exc}$ are respected. The allocation difficulty for a problem is given by these percentages.

Several classes of problems have been defined depending on the difficulty of both allocation and schedulability problems. Table 4 describes the parameters of each basic difficulty class. By combining them, categories of problems can be specified. For instance, a W-X-Y-Z category corresponds to problems with a memory difficulty in class W, an allocation difficulty in class X, a schedulability difficulty in class Y and a network difficulty in class Z.

Memory		Allocation				Schedulability		Message		
	$\%_{mem}$		$\%_{res}$	$\%_{co}$	$\%_{exc}$		$\%_{global}$		mes	$\%_{mes}$
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

Table 4
Details of difficulty classes

The method with the global constraint is denoted as Global-CPRTA and the Benders decomposition method is denoted by Benders-CPRTA. A problem with a solution is called a *consistent* problem. A problem for which the proof that no solution exists is given, is called an *inconsistent* problem. A problem for which we cannot state whether a solution exists or not, is an *open* problem. A time limit is fixed at 10 minutes per problem. If the problem is still open within this time limit, the search is stopped.

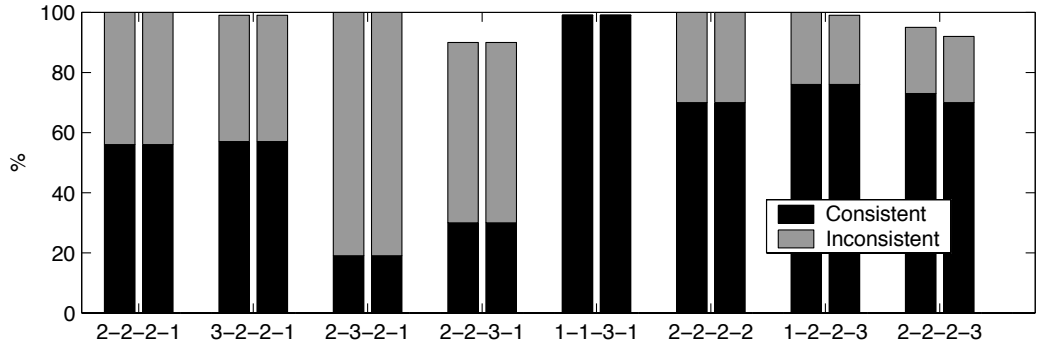

Fig. 6. Sum of percentages (y-axis) of consistent (black) and inconsistent (grey) problems for each difficulty class (x-axis) with Benders-CPRTA (right) and Global-CPRTA (left)

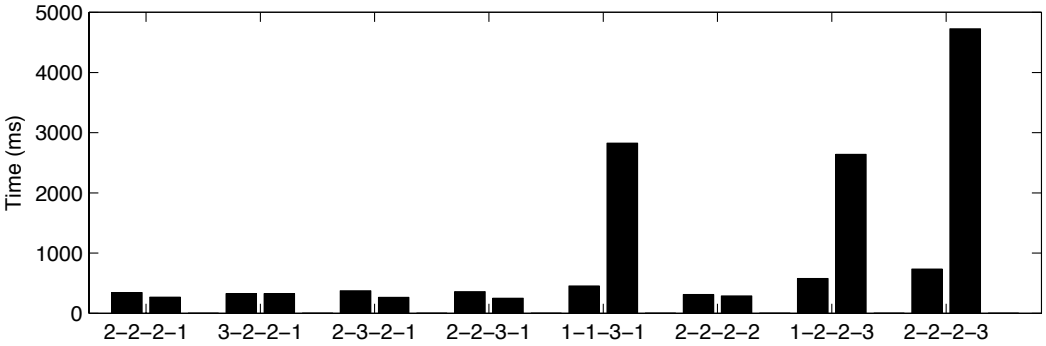

Fig. 7. Median time (y-axis) for solving a problem (consistent and inconsistent) for each difficulty class (x-axis) with Benders-CPRTA (right) and Global-CPRTA (left)

Figures 6 and 7 summarize some of the results (Appendix 2 gives detailed results). We do not give the results for all the intermediate classes of problems (like 1-1-1-1, 2-1-1-1, etc.) because they are easily solved and they do not exhibit a specific behavior. The data are obtained on 100 instances (40 tasks, 7 processors) per class of difficulty with a Pentium 4 (3.2 GHz).

Figure 6 gives the percentage of consistent and inconsistent problems solved by Benders-CPRTA and Global-CPRTA. For a difficulty class, Global-CPRTA is the left bar and Benders-CPRTA is the right one. Benders- and Global-CPRTA

are efficient for solving problems with less than 20% of open problems for each difficulty class (and less than 10% if we do not consider 2-2-3-1).

Figure 7 gives the median time for solving a problem. For hard schedulabilty problems (*e.g.* class 1-1-3-1) and for a heavy load network (*e.g.* classes 1-2-2-3 and 2-2-2-3), Benders-CPRTA is clearly slower than Global-CPRTA. We can explain this because, for Benders-CPRTA, the schedulability constraint set is empty at the beginning of the search. Therefore, all the knowledge dealing with schedulability has to be learnt from the subproblem. Furthermore, learning is only effective when a valid solution is produced by the master problem solver and, as a consequence, it is not really integrated into the cp algorithm. Global-CPRTA improves performances from this point of view. For a heavy loaded network, Benders-CPRTA has the same difficulty in taking into account the schedulability of messages. Moreover, we have observed that nogoods inferred from message unschedulability are usually "weaker" (the search space cut is smaller) than those inferred from task unschedulability. Learning is thus less efficient for this kind of problem.

However, Benders-CPRTA is on average better in time than Global-CPRTA for classes where schedulability constraints are negligible. In these cases, for Benders-CPRTA, memory and allocation constraints are directly considered by the master problem, which does not consider schedulability, contrary to Global-CPRTA, and thus problems are solved faster.

8.2 Comparison with simulated annealing

As stated in Section 3, there are numerous search methods for static allocation problems. However, each technique needs some expertise for optimizing it when a new model is considered. For example, for the branch-and-bound technique, branching and bounding algorithms need to be implemented for each specific allocation problem, and the efficiency of the method depends on the effectiveness of them. Moreover, each technique is sensitive to some parameters. For example, a branch-and-bound is sensitive to the enumeration order of variables. Thus, implementing an efficient solving algorithm (with optimization and smart parameters) is a full problem for each method. Establishing a comparison between different techniques is beyond our present concern.

Thus, we have only implemented and optimized a simulated annealing algorithm (SA) for comparison purposes. The SA algorithm is inspired by [63]. This choice was motivated by similarity with our model and because this work is often referred to allocation problems of hard real-time systems. In [63], the energy function takes into account residence, exclusion and memory constraints as well as task deadline constraints. To be consistent with the CPRTA model,

the schedulability of messages on the CAN bus and co-residence constraints have been integrated in the same way. All constraints are weighted to make the cost of each one uniform in the objective function. We took great care in its implementation so as to reduce the computation time of this energy function. Moreover, different neighbor functions were experimented as well as different parameters like cooling, number of iterations and initial temperature so as to optimize its behavior. Figures 8 and 9 show only the best results computed by SA for all difficulty classes. When an iteration limit is reached, a new initial solution is produced and a new research is done until a time limit fixed at 10 minutes. If this time limit is reached, the problem is considered as open.

Notice that SA is a heuristic method. As a consequence, in our case, SA can only conclude on consistent problems (Fig. 8). SA is always as or less efficient than Global-CPRTA. Figure 9 shows the median time to solve a consistent problem with SA and CPRTA. Global-CPRTA is always faster than SA. Benders-CPRTA is also faster than SA except for difficult schedulability problems 1-1-3-1. All these results show that cp is an efficient way to mix different constraints without an initial step.

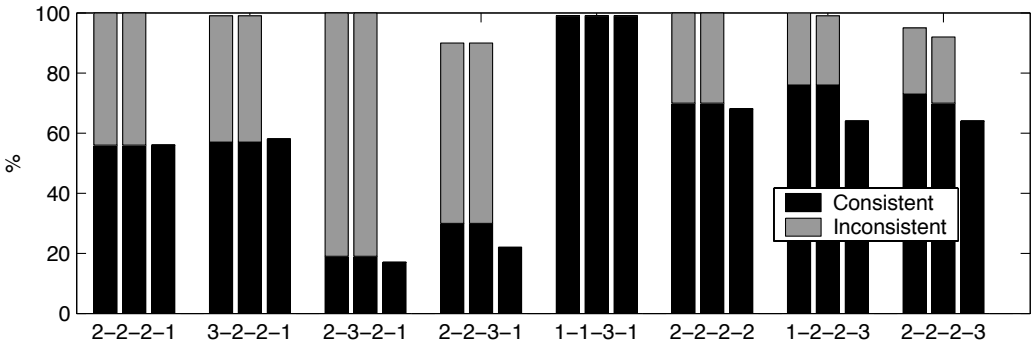

Fig. 8. Sum of percentages (y-axis) of consistent (black) and inconsistent (grey) problems for each difficulty class (x-axis) with SA (right), Benders-CPRTA (middle) and Global-CPRTA (left)

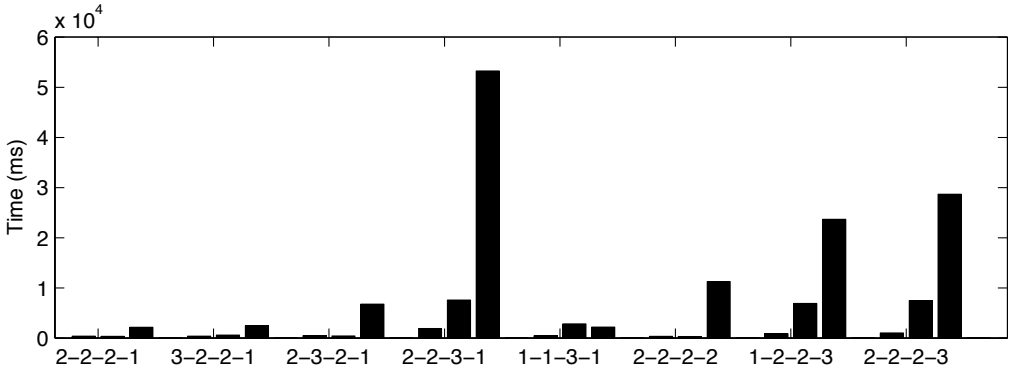

Fig. 9. Median time (y-axis) for solving a problem (consistent and inconsistent) for each difficulty class (x-axis) with SA (right), Benders-CPRTA (middle) and Global-CPRTA (left)

9 Explanations

In comparison with other search methods, using a constraint solver may help "intrinsically" to answer some classical queries when a problem is proved without a solution such as: why does my problem have no solution? Usually, when the domain of a variable of a csp becomes empty (no value exists that will respect all the constraints on that variable), basic cp systems notify the user that there is no solution. Nevertheless, thanks to the versatility of the explanation-based constraint approach we use, those relevant constraints, which explain the failure, are made available [30]. This advantage of the approach is another motivation to go further in the utilization of cp. At present, only Benders-CPRTA has been investigated with this objective.

Thus, in the case of an allocation problem for which no solution has been found, we analyze the set of constraints that is returned to explain the problem inconsistency. There can be many reasons to explain inconsistency. At the design level, we would like to be able to incriminate high level characteristics of the system such as: allocation constraints, schedulability requirements of tasks, processors or network limitation. However, two points of view, based on the software or hardware architecture, can be adopted. To begin, we only focus on the characteristics of the software architecture by analyzing how each task is "responsible" for the failure. We give some insight into the way a critical task from the schedulability point of view can be identified. Each failure of the search process due to schedulability is analyzed and transformed into a constraint criterion that encapsulates an accurate reason for this failure. The study of these criteria may lead to the guilty task(s). The rationale of this evaluation is based on the following remarks:

- The more a task appears within a nogood, the more this task has an impact on the schedulability inconsistency.
- The level of propagation performed by a nogood (either *NotAllEqual*(x_i) or $\sum w_{ij} < B$), *i.e* its impact within the proof, is strongly related to its size (the number of tasks it involves). "Small" *NotAllEqual* have greater impact.

In its general form, a constraint (learnt from a nogood) is defined by *NotAllEqual*(x_i) or $\sum w_{ij} < B$ (see Section 7.2). We denote `NAE` the set of constraints in the *NotAllEqual* form and `SUM` the set of constraints in the second form. For a task τ_i a constraint criterion $\mathcal{C}_i$ is evaluated:

$$\mathcal{C}_i = \sum_{\substack{c \in \texttt{NAE} \\ x_i \in c}} \frac{1}{\#c} + \sum_{\substack{c \in \texttt{SUM} \\ \exists j, w_{ij} \in c \vee w_{ji} \in c}} \frac{1}{\#c} \tag{15}$$

This criterion considers the presence of a task in each constraint and its im-

pact. The bigger $\mathcal{C}_i$ is, the bigger the impact of τ_i is on the inconsistency. By studying tasks with high $\mathcal{C}_i$ and understanding why they have such an impact on the inconsistency (*e.g.* low priority allocation, too large processor utilization), it is possible to change some requirements (*e.g.* by adapting priorities, or choosing a different version for a task with another period) and so to obtain a solution for the problem.

τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$	τ_i	$\mathcal{C}_i$
τ_{19}	6.33	τ_{13}	4.78	τ_2	3.22	τ_3	2.53
τ_{14}	5.98	τ_9	3.95	τ_1	2.85	τ_{16}	2.25
τ_{11}	5.98	τ_6	3.83	τ_{10}	2.77	τ_{18}	1.97
τ_5	5.42	τ_7	3.45	τ_4	2.65	τ_8	1.73
τ_{12}	5.42	τ_{15}	3.32	τ_{17}	2.55	τ_0	1.15

Table 5
Constraint criteria computed on the example of Section 7.3

Table 5 gives $\mathcal{C}_i$ computed with Œdipe in accordance with equation (15) for the inconsistent example of Section 7.3 with Œdipe [13]. Task τ_{19} has the biggest $\mathcal{C}_i$. This task has a low priority together with a high processor utilization ($C_{19}/T_{19} = 0.32$). By just changing its priority to the highest one, and reusing Benders-CPRTA, we found a solution for this problem.

Notice that this process consists in analyzing the final set of constraints with a heuristic based on the information about nogoods gathered during the search with Benders decomposition. This process can be generalized to memory and allocation constraints by the use of a specific search technique [48]. However, this work is in progress and still needs more study before becoming significant.

10 Conclusion and future work

In this paper, we present an original and complete approach (CPRTA) to solve a hard real-time allocation problem with constraint programming. To tackle this problem, two approaches are proposed. For the first one, timing constraints have been conducted to define a global constraint. This method has been optimized to speed up the search by exploiting specific properties of schedulability analysis. For the second one, a decomposition method built on a logic Benders scheme is used. The whole problem is split into a master problem handling allocation and resource constraints and a subproblem for timing constraints. A rich interaction between master and subproblems is performed with the computation of minimal sets of unschedulable tasks and messages. It implements a learning technique in an effort to combine the various issues into a solution that satisfies all the constraints.

Experimental results show that these two methods produce an efficient way of solving allocation problems. Thanks to the specialized global constraint, the Global-CPRTA achieves a better performance.

Another important specificity of CPRTA is its completeness, *i.e.* if a problem has no solution, the search algorithm is able to prove it. In future work, our aim is to integrate into the design process an intelligent tool based on CPRTA able to return pertinent explanations justifying the failure.

In the future, we hope to improve the solving method by considering properties of schedulability tests in a constraint expressed by automata [51]. The problem of priorities assignment is another way to investigate by using the optimal algorithm of Audsley [5] in the Benders decomposition or a new global constraint.

11 Acknowledgements

Many thanks to Carol Robins for her excellent work in making the document more readable.

References

[1] T. F. Abdelzaher and K. G. Shin. Period-based partitioning and assignment for large real-time applications. *IEEE Transactions on Computers*, 49(1):81–87, 2000.

[2] J. Aguilar and E. Gelenbe. Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences*, 97(2):199–219, 1997.

[3] S. Ali, J.-K. Kim, H. Siegel, A. Maciejewski, Y. Yu, S. Gundala, S. Gertphol, and V. Prasanna. Greedy heuristic for resource allocation in dynamic distributed real-time heterogeneous computing systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002)*, volume 29, 2002.

[4] P. Altenbernd and H. Hansson. The slack method: A new method for static allocation of hard real-time tasks. *Real-Time Systems*, 15(2):103–130, 1998.

[5] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292, 1993.

[6] L. Baccouche. *Un Mécanisme d'Ordonnancement Distribué de Tâches Temps Réel*. PhD thesis, Institut National Polytechnique de Grenoble, 1995.

[7] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[8] T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to Benders decomposition: a case study. *Lecture Notes in Computer Science*, 2470:603–617, 2002.

[9] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30:129–154, 2005.

[10] Bosch. *CAN Specification version 2.0*, 1991.

[11] A. Burns, M. Nicholson, K. Tindell, and N. Zhang. Allocating and scheduling hard real-time task on a point-to-point distributed system. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, pages 11–20, 1993.

[12] H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a hard real-time task allocating problem. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP 2004)*, 2004.

[13] H. Cambazard and P.E. Hladik. ŒDIPE. http://oedipe.rts-software.org/index.php.

[14] M. Coli and P. Palazzari. A new method for optimisation of allocation and scheduling in real-time applications. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, pages 262–269, 1995.

[15] M. DiNatale and J. A. Stankovic. Applying of simulated annealing methods to real-time scheduling and jitter control. In *proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS 1995)*, 1995.

[16] C. Ekelin. *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. PhD thesis, Chalmers University of Technology, 2004.

[17] A. A. Elsadek and B. E. Wells. A heuristic model for task allocation in heterogeneous distributed computing systems. *The International Journal of Computers and Their Applications*, 6(1), 1999.

[18] F. Ercal, J. Ramanujan, and P. Sadayappan. Task allocation on a hyper-cube by recursive bipartitioning. *Journal of Parallel and Distributed Computing*, 10:35–44, 1990.

[19] 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. In *Proceedings of the 3th World Multiconference on Systemics Cybernetics and Informatics*, pages 575–577, 1999.

[20] E. Ferro, D. Sanchez, R. Cayssials, and J. Orozco. New scheduling and assignment real time control task with precedence and deadline constraint in distributed control systems, 2000. url:citeseer.ist.psu.edu/405977.html.

[21] J. Fredriksson, K. Sandstrm, and M. Åkerholm. Optimizing resource usage in component-based real-time systems. In *Proceedings of the 8th International Symposium on Component-based Software Engineering (CBSE8)*, 2005.

[22] J. Goossens and C. Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the RTS Embedded System (RTS'01)*, pages 133–147, 2001.

[23] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien. Dynamic constraint programming for solving hard real-time allocation problems. Technical Report 7, IRCCyN, 2005.

[24] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien. How to solve allocation problems with constraint programming. In *Proc. of the Work In Progress of the 17th Euromicro*, pages 25–28, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[25] P.-E. Hladik and A.-M. Déplanche. Extension au réseau can des problèmes de placement. Technical Report 4, IRCCyN, 2005.

[26] J. Hooker, G. Ottosson, E. Thorsteinsson, and H. Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review*, 15(1):11–30, 2000.

[27] J. N. Hooker and G. Ottoson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[28] V. 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.

[29] U. Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *Proc. of the 8th International Joint Conference on Artificial Intelligence (IJCAI 01)*, 2001.

[30] Narendra Jussien. The versatility of using explanations within constraint programming. Habilitation thesis of Université de Nantes, 2003.

[31] Choco. http://choco.sourceforge.net/.

[32] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou. An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures. In *Proceedings of the 8th EuroPDP*, pages 406–413, 2000.

[33] F. Laburthe. Choco: implementing a cp kernel. In *Proceedings of CP 00 Post Conference Workshop on Techniques for Implementing Constraint Programming Systems*, 2000.

[34] E. Lawler. Recent results in the theory of machine scheduling. *Mathematical Programming: The State of the Art*, 1983.

[35] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS 1990)*, pages 201–209, 1990.

[36] R. Lepère and D. Trystram. A new clustering algorithm for scheduling task graphs with large communication delays. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002*, 2002.

[37] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 22:237–250, 1982.

[38] V. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on computers*, 37(11):1384–1397, 1988.

[39] Y. Monnier, J.-P. Beauvais, and A.-M. Déplanche. A genetic algorithm for scheduling tasks in a real-time distributed system. In *Proceedings of the 24th Euromicro Conference*, 1998.

[40] T. Muntean and E-G. Talbi. Hill-climbing, simulated annealing and genetic algorithms, a comparative study. In *Proceedings of the 26th Hawaii International Conference on Task Scheduling in Parallel and Distributed Systems (HICSS-26)*, 1993.

[41] 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.

[42] J. Oh, H. Bahn, C. Wu, and K. Koh. Pareto-based soft real-time task scheduling in multiprocessor systems. In *Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC'00)*, pages 24–28, 2000.

[43] OSEK Group. *OSEK/VDX Communication version 3.0.2*.

[44] H-J. Park and B. Kim. An optimal scheduling algorithm for minimizing the computing period of cyclic synchronous tasks on multiprocessors. *The Journal of Systems and Software*, 56:213–229, 2001.

[45] D-T. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12), 1997.

[46] S. C. S. Porto and C.C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High-Speed Computing*, 7(2), 1993.

[47] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS 1990)*, 1990.

[48] P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP 2004)*, 2004.

[49] J.C. Régin. *Constraints and Integer Programming Combined*, chapter Global Constraints and Filtering Algorithms. Kluwer, 2003.

[50] M. Richard, P. Richard, and F. Cottet. Allocating and scheduling tasks in multiple fieldbus real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 16, pages 137–144, 2003.

[51] G. Richaud, H. Cambazard, B. O'Sullivan, and N. Jussien. Automata for nogood recording in constraint satisfaction problems. In *CP06 Workshop on the Integration of SAT and CP techniques*, Nantes, France, Sep 2006.

[52] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[53] F. Sandnes. A hybrid genetic algorithm applied to automatic parallel controller code generation. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, pages 70–75, 1996.

[54] J. Santos, E. Ferro, J. Orozco, and R. Cassials. A heuristic approach to multitask-multiprocessing assignment problem using the empty-slots method and rate monotonic scheduling. *Real-Time Systems*, 13(2):167–199, 1997.

[55] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4):335–357, 2000.

[56] M. Silva, C. Cardeira, and Z. Mammeri. Solving real-time scheduling problems with hopfield-type neural networks. In *Proceedings of the Euromicro Conference*, pages 671–678, 1997.

[57] H. Stone. Multiprocessor scheduling the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, 3(1):85–93, 1977.

[58] R. Szymanek, F. Gruian, and K. Kuchcinski. Digital systems design using constraint logic programming. In *Proceedings of The Practical Application of Constraint Technologies and Logic Programming (PACLP 2000)*, 2000.

[59] R. Szymanek and K. Kuchcinski. Partial task assignment of task graphs under heterogeneous resource constraints. In *Proceedings of the 40th conference on Design Automation (DAC '03)*, pages 244–249, 2003.

[60] E-G. Talbi and T. Muntean. General heuristics for the mapping problem. In *Proceedings of the World Transputer Conference*, 1993.

[61] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. *Lecture notes in Computer Science*, 2239, 2001.

[62] K. Tindell, H. Hansson, and A. Wellings. Analysis real-time communications: controller area network (CAN). In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS 1994)*, pages 259–265, 1994.

[63] K. W. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[64] L. Vargas and R. Olivaira. Empirical study of tabu search, simulated annealing and multi-start in fieldbus scheduling. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, volume 1, pages 101–108, 2005.

[65] S. Wang, J. Merrick, and K. Shin. Component allocation with multiple resource constraints for large embedded real-time software design. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, 2004.

[66] C. Wong, F. Thoen, K. Catthoor, and D. Verkest. Requirements for static task scheduling in real-time embedded systems. In *Proceedings of the 3rd Workshop on System Design Automation (SDA 2000)*, pages 23–30, 2000.

Appendix 1: Properties of the schedulability analysis

Property 11.1 *If a partial allocation a is unschedulable, then all decisions from a produce an unschedulable allocation.*

Proof: Consider Eq.8 that defines $R_i(a)$, its computation is derived by iteratively calculating formula $R_i^{(n)}(a) = C_i + \sum_{\tau_j \in hp_i(a)} \left\lceil \frac{R_i^{(n-1)}(a)}{T_j} \right\rceil C_j$ with $R_i^{(0)}(a) = C_i + \sum_{\tau_j \in hp_i(a)} C_j$.

Property is proved if for a task $\tau_i \in \mathcal{U}$, all decisions δ from a are such that $R_i(a') > T_i$ with $a' = a + \delta$. It is easy to prove it by induction by pointing out that $hp_i(a) \subset hp_i(a')$.

Induction basis. By definition $R_i^{(0)}(a') \geqslant R_i^{(0)}(a)$.

Induction step. Assume that $R_i^{(n)}(a') \geqslant R_i^{(n)}(a)$ is true. Therefore,

$$\begin{aligned} R_i^{(n+1)}(a') &\geqslant C_i + \sum_{\tau_j \in hp_i(a')} \left\lceil \frac{R_i^{(n)}(a)}{T_j} \right\rceil C_j \\ &= C_i + \sum_{\tau_j \in hp_i(a)} \left\lceil \frac{R_i^{(n)}(a)}{T_j} \right\rceil C_j + \sum_{\tau_j \in hp_i(\delta)} \left\lceil \frac{R_i^{(n)}(a)}{T_j} \right\rceil C_j \\ &\geqslant R_i^{(n+1)}(a) \end{aligned}$$

At the fix-point, we have : $R_i(a') \geqslant R_i(a) > T_i$.

The same reasoning could be made for messages by considering Eq.11 and Eq.12. For a messsage M_{ij} we have $hp_{ij}(a) \subset hp_{ij}(a')$ and $lp_{ij}(a) \subset lp_{ij}(a')$ □

Property 11.2 *Consider a schedulable partial allocation a and δ, a decision from a that allocates only τ_i with $\delta(\tau_i) = p_k$. If τ_i is unschedulable in $a' = a+\delta$, then all decisions from a that allocate a task τ_j with $prio_j < prio_i \wedge C_j \geq C_i \wedge T_j \leq T_i$ on p_k produce unschedulable allocations.*

Proof: Consider a partial schedulable allocation a and two others a' and a'' such that $a' = a + \delta_1$, $a'' = a + \delta_2$, $\delta_1(\tau_i) = \delta_2(\tau_j) = p_k$, $R_i(a') > T_i$ and $prio_j < prio_i \wedge C_j \geq C_i \wedge T_j \leq T_i$. We want to prove that $R_j(a'') > T_j$.

By definition of priorities, $hp_j(a'') \supseteq hp_i(a')$.

The property will be proved by induction.

Induction basis. $R_i^{(0)}(a') = C_i + \sum_{\tau_x \in hp_i(a')} C_x \leqslant C_j + \sum_{\tau \in hp_j(a'')} C = R_j^{(0)}(a'')$.

Induction step. Assume that $R_j^{(n)}(a'') \geqslant R_i^{(n)}(a')$ is true. Therefore,

$$
\begin{aligned}
R_j^{(n+1)}(a'') &\geqslant C_i + \sum_{\tau_x \in hp_j(a'')} \left\lceil \frac{R_i^{(n)}(a')}{T} \right\rceil C_x \\
&= C_i + \sum_{\tau_x \in hp_i(a')} \left\lceil \frac{R_i^{(n)}(a')}{T} \right\rceil C_x + \sum_{\tau_x \in hp_j(a'') - hp_i(a')} \left\lceil \frac{R_i^{(n)}(a')}{T} \right\rceil C_x \\
&\geqslant R_i^{(n+1)}(a')
\end{aligned}
$$

At the fix-point, we have : $R_j(a'') \geqslant R_i(a') > T_i \geqslant T_j$ $\square$

Property 11.3 *Consider a schedulable partial allocation a and δ, a decision from a that allocates only τ_i with $\delta(\tau_i) = p_k$ and where τ_b is unschedulable due to τ_i, then all decisions from a that allocate a task τ_j with $prio_j > prio_b \wedge C_j \geq C_i \wedge T_j \leq T_i$ on p_k produce unschedulable allocations.*

Proof: Consider a partial allocation a, two others a' and a'' such that $a' = a + \delta_1$, $a'' = a + \delta_2$, $\delta_1(\tau_i) = \delta_2(\tau_j) = \delta_1(\tau_b) = \delta_2(\tau_b) = p_k$, $R_b(a') > T_b$ and $prio_j > prio_b \wedge C_j \geq C_i \wedge T_j \leq T_i$. We want to prove that $R_b(a'') > T_b$.

By definition of priorities, $hp_b(a'') - \{\tau_j\} = hp_b(a') - \{\tau_i\} = hp_b$.

The property will be proved by induction.

Induction basis. $R_b^{(0)}(a') = C_b + \sum_{\tau \in hp_b} C_j + C_i \leqslant C_b + \sum_{\tau \in hp_b} C + C_j = R_b'^{(0)}(a'')$.

Induction step. Assume that $R_b^{(n)}(a'') \geqslant R_b^{(n)}(a')$ is true. Therefore,

$$R_b^{(n+1)}(a'') \geqslant C_b + \sum_{\tau \in hp_b} \left\lceil \frac{R_b^{(n)}(a')}{T} \right\rceil C + \left\lceil \frac{R_b^{(n)}(a')}{T_i} \right\rceil C_i$$
$$\geqslant R_b^{(n+1)}(a')$$

At the fix-point, we have : $R_b(a'') \geqslant R_b(a') > T_b$ □

Appendix 2: Experimental results

The following tables are the detailed forms of the results shown in Section 8. CPU is the computation time in milli-seconds. NODE is the number of nodes explored during the search with the Global-CPRTA. ITER is the number of iterations between the master problem and the subproblem for the Benders-CPRTA. NOG is the mean of nogoods inferred from the subproblem for the Benders-CPRTA. The data are obtained on 100 instances (40 tasks, 7 processors) per class of difficulty with a Pentium 4 (3.2 GHz).

	CPU	NODE	CPU	NODE	CPU	NODE
2-2-2-1	100 solved		56 consistent		44 inconsistent	
Median	343.0	32.0	359.5	35.0	297.0	1.0
3-2-2-1	99 solved		57 consistent		42 inconsistent	
Median	328.0	30.0	359.0	34.0	296.5	1.0
2-3-2-1	100 solved		19 consistent		81 inconsistent	
Median	375.0	1.0	437.0	25.0	344.0	1.0
2-2-3-1	90 solved		30 consistent		60 inconsistent	
Median	359.0	1.0	1906.5	604.5	328.0	1.0
1-1-3-1	99 solved		99 consistent		0 inconsistent	
Median	453.0	78.0	453.0	78.0	0.0	0.0
2-2-2-2	100 solved		70 consistent		30 inconsistent	
Median	312.0	30.0	328.0	33.0	266.0	1.0
1-2-2-3	100 solved		76 consistent		24 inconsistent	
Median	578.0	72.0	851.5	160.0	297.0	1.0
2-2-2-3	95 solved		73 consistent		22 inconsistent	
Median	734.0	120.0	1016.0	166.0	289.5	1.0

Table 6
Results for Global-CPRTA

	CPU	ITER	NOG	CPU	ITER	NOG	CPU	ITER	NOG
2-2-2-1	100 solved			56 consistent			44 inconsistent		
Median	266.0	12.0	44.0	343.5	18.5	83.5	203.0	1.0	13.0
3-2-2-1	99 solved			57 consistent			42 inconsistent		
Median	328.0	20.0	61.0	594.0	42.0	147.0	219.0	1.0	11.0
2-3-2-1	100 solved			19 consistent			81 inconsistent		
Median	265.0	1.0	14.0	422.0	29.0	111.0	250.0	1.0	13.0
2-2-3-1	90 solved			30 consistent			60 inconsistent		
Median	250.0	2.0	22.5	7586.0	80.0	383.0	219.0	1.0	15.5
1-1-3-1	99 solved			99 consistent			0 inconsistent		
Median	2828.0	78.0	367.0	2828.0	78.0	367.0	0.0	0.0	0.0
2-2-2-2	100 solved			70 consistent			30 inconsistent		
Median	289.5	12.0	40.0	313.0	18.0	60.0	226.5	1.0	9.5
1-2-2-3	99 solved			76 consistent			23 inconsistent		
Median	2641.0	152.0	375.0	6891.0	302.5	938.0	281.0	1.0	15.0
2-2-2-3	92 solved			70 consistent			22 inconsistent		
Median	4726.5	179.5	452.5	7476.5	289.0	859.5	265.0	1.0	11.5

Table 7
Results for Benders-CPRTA

Class	**2-2-2-1**	**3-2-2-1**	**2-3-2-1**	**2-2-3-1**	**1-1-3-1**	**2-2-2-2**	**1-2-2-3**	**2-2-2-3**
% consistent	56.0	58.0	17.0	22.0	99.0	68.0	64.0	64.0
Median time	2140.0	2438.0	6750.0	53234.5	2187.0	11273.0	23664.0	28664.0

Table 8
Results for SA