Allocating Hard Real-Time Tasks with Constraint Programming

Pierre-Emmanuel Hladik [a,*], 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 an 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 new approaches are proposed for solving these kinds of problems which produce in their current version as acceptable performances as classical algorithms do. Some experimental results are given to show it. Moreover, CPRTA exhibits very interesting properties. It is complete — *i.e.*, if a problem has no solution — the algorithm is able to prove it ; it is non-parametric — *i.e.*, it does not require specific initializations — 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

* Corresponding author. Tel.: 00 33(0)240-3769-21 Fax: 00 33(0)240-3769-30
Email address: `Pierre-Emmanuel.Hladik@irccyn.ec-nantes.fr` (Pierre-Emmanuel Hladik).

1 Introduction

Real-time systems have applications in many industrial areas: telecommunication systems, automotive, aircraft, robotics, etc. Today's applications are becoming more and more complex, as much in their software part (an increasing number of concurrent tasks with various interaction schemes), as in their execution platform (many distributed processing units interconnected through specialized network(s)), and in their numerous functional and non-functional requirements too (timing, resource, power, etc. constraints). 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. In general, it is a difficult constraint satisfaction problem. Even if it has to be solved off-line most of the time, it needs efficient and adaptable search techniques which are able to be integrated into a more global design process. Furthermore, it is desirable that those techniques return relevant information intended to help the designer who is faced with architectural choices. The "binary" result, in particular, (has a feasible allocation been found?: yes and here it is, or no, and that's all) which is usually returned by the search algorithm is not satisfactory in failure situations. The designer would expect some explanations justifying the failure and enabling him to revisit his design. Therefore, more sophisticated search techniques that would be able to collect some knowledge about the problem they solve are required. Here 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, preemptive tasks to distributed processors in the context of fixed priority scheduling, to respect schedulability but also to account for requirements related to memory capacity, co-residence, redundancy, and so on. We assume that the characteristics of tasks (execution time, priority, etc.) and the ones of the physical architecture (processors and network) are all known a priori [1].

Assigning a set of hard preemptive real-time tasks in a distributed system under allocation and resource constraints is known to be an NP-Hard problem [31]. Up to now, it has been massively tackled with heuristic methods [48,43], simulated annealing [56,10] and genetic algorithms [20,47]. Recently, Szymanek et al. [52] and especially Ekelin [17] have used constraint programming to produce an assignment and a pre-runtime scheduling of distributed systems under optimization criteria. Even if their context is different from ours, their results have shown the ability of such an innovative approach to solve an allocation problem for embedded systems and have encouraged us to go further.

[1] Only static real-time systems are here considered.

In this paper, two approaches are considered. The first one introduces a global constraint[2] and an *ad hoc* algorithm, *i.e.*, a filtering algorithm, to tackle schedulability. This algorithm is a custom-written filtering algorithm designed in order 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 [55,26,24,9]. It is a decomposition-based method (related to logic Benders-based decomposition [25]) 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 mistakes.* Lastly we underline that a fundamental property of these methods is their completeness: when a problem has no solution, it is able to prove it (contrary to heuristic methods that are unable to decide).

The remainder of this paper is organized as follows. In section 2, we describe the problem. Section 4 introduces shortly the constraint programming before translate the problem as a constraint satisfaction one in Section 5. Aftermentioned, the two approaches 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 The problem description

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

[2] A constraint is said global when it is a conjunction of a set of constraints.

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, preempting eventually a lower priority task. Those 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 as far as its timing behaviour (including its protocol rules) is predictable. Thus the first experiments we have conducted addressed a token ring network.

CAN (Controller Area Network) [11] is both a protocol and physical network. CAN works as a broadcast bus meaning that all connected nodes will be able to 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 gets selected to transfer, the message with the highest priority, waiting on any connected node, will get transmitted first. When at least one bit of a message has started to be transfered it can't get preempted even though higher priority messages arrive. As a result, the CAN's behaviour will be seen subsequently as the one of a non preemptive fixed priority message scheduling.

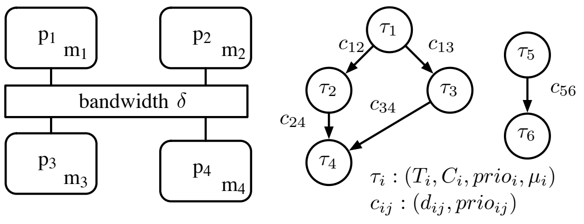

Fig. 1. An example of hardware (left) and software (right) architecture.

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 which 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 preemption 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 $prio_i < prio_j$. 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) [3]. In this model, we assume that communicating tasks have

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

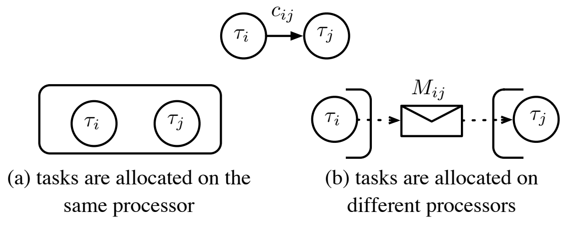
c_{ij}
τ_i
τ_j
M_{ij}
(a) tasks are allocated on the same processor
(b) tasks are allocated on different processors

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

the same activation period. But we don't 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 from OSEK-COM specifications [39]. OSEK-COM is an uniform communication environment for automotive control unit application software. It defines common software communication interface and behaviour for internal communications (within an electronic control unit) and external ones (between networked vehicle nodes) which is 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 with the *periodic transmission mode* of OSEK-COM. In this mode data production and message transmission aren't synchronised: 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 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 proper message M_{ij}. Moreover in this paper, for a 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 with the M_{ij} message arrival: 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 [23] 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 the mapping A which respects the whole set of constraints described in the immediate below.

2.2.1 Resource constraints

Three kinds of constraints are considered [4]:

- **Memory capacity**: The memory use of a processor p_k cannot not 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} stands for the message size ; it is a function of d_{ij} depending of the message structure (see below the Section 2.2.3 about the message worst-case response time for its computation).

[4] Precise units aren't specified but obviously they have to be consistent with the given expressions

2.2.2 Allocation constraints

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

- **Residence**: a task may need 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). This constraint is 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**: This constraint enforces several tasks 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 placed 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

They are expressed by the means of 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 instanciated 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 [32]: S is schedulable if and only if, for each task and message of S, its worst-case response time is less 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 in the immediate following.

Task worst-case response time. For independent and periodic tasks with a preemptive 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 which 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 [32]:

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

where $\lceil x \rceil$ is the smallest integer $\geq x$ 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 number of times tasks with higher priority will execute 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-preemptive fixed priority scheduling strategy. Thus when doing a 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 preempted during its first transmitted bit instead of its whole execution time. Second a blocking time, i.e. the largest 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 [57]:

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

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 in relation with the bus bandwidth δ, τ_{bit} (second) $= \frac{1}{\delta \text{ (bit per second)}}$) ; 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 [57]:

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

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

3 Related work

A mapping of a real-time system 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 a classification is difficult because of various natures of problems.

[7,17] give a classification for mapping problem. A problem is specified according to the hardware model (multiprocessor, distributed, homogeneous, etc.), the software model (periodic tasks, deadline, precedence relations, etc.), the constraints (memory, allocation constraints, etc.) and the objective functions (minimise execution and communication costs, load balancing, etc.). However, to have a complete description of a problem, it is necessary to consider the implementation too. To implemented an allocation various strategies could be used. Yeckle and Rivera propose in [61] a taxonomy of implementation of mapping strategies. They begin to distinguish static and dynamic mapping.

For a *dynamic mapping*, the allocating or/and the scheduling are done in-line. In this case, a mechanism has to make decisions to choose during the execution of the application the allocating and the scheduling. Different classes of mechanisms exist: distributed and non-distributed. *Distributed allocation* means that the mechanism is physically distributed through processors. Contrary, a *non-distributed allocation* has a centralised mechanism that decide mapping and scheduling for all the system.

For a *static mapping*, the allocating and the scheduling are assumed to be known before the start of the application and never change. All task of a job is executed always on the same processor. For static mapping, various scheduling policies exist: cycle scheduling, fixed priority, dynamic priority, round-robin, etc. All parameters for scheduling must be fixed before the execution of the application.

Many research efforts in the literature concentrate on mapping real-time applications and numerous search techniques have been studied. Since the problem of allocating tasks is generally NP-hard, some form of enumerative methods or approximation using heuristics needs to be developed for this problem: The graph theoretic techniques [51,34,19]; The branch-and-bound [37,41,40,46,59]; The genetic algorithm [36,5,47,2,35,21,38,22]; The clustering [4,1,33]; The steepest descent (or Hill climbing) [36,58]; The tabu search [42,58]; The simulated annealing [56,12,36,15,16,18,58]; The neuronal network [50,2]; Some dedicated heuristics [48,18,29,60,3].

The major ways in which our work differs from the works cited above are:

(1) Our research focuses on fixed priority scheduler scheduler and not to produce an off-line schedulin [].
(2) Our objective is to validate an application (find a solution) and not optimise a function as the workload balancing [54,2], the number of used processors [38] or the response time of tasks [41,2]. Our approach can integrate easily an optimisation function, but we don't consider this case from reason of simplicity.
(3) Our method does not depend of some classes of constraints but it is a general scheme from solving mapping problem.

Research efforts in [56,12,37,48,4,46,22] do consider task assignments with fixed priority scheduler. Allocation and scheduling are usually considered as two independent stages. In many works, the scheduling policy is a priori known, as in [56,12,37,48,4,22]. These approachs mainly focus on the allocation process. In [46] Richard *et al.* propose a method that simultaneously allocates tasks to processors and assigns priorities to tasks and messages. 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 algorithm can be used, however, our approach does not consider a specific scheduling policy.

We based our research on the model in [56] with few differences as the communication protocol and the schedulability test. In [56,12] the mapping problem is solved by simulated annealing. However, simulated annealing is very sensitive to its initial parameters, *i.e.*, temperature, cooling and the termination criterion. It is very difficult to find parameters which fit well for all applications. Moreover, the objective function depends of different constraint and must be redefined if new constraints are added.

In the research effort [37,46], a branch-and-bound search algorithm was used for the mapping problem. Contrary to [46], in [37] a large variety of constraints are assumed, *e.g.* group of tasks, memory capacity. Branch-and-bound is an optimal search algorithm but the main its drawback is that there is no universal bounding algorithm. Therefore, the branching and bounding algorithms that are specially designed for each mapping problem. Moreover, the efficiency of the method depends on the effectiveness of the branching and bounding algorithms used: bad choices could lead to repeated branching, without any pruning, until the subregions become very small. In that case the method would be reduced to an exhaustive enumeration of the domain, which is often impractically large. So it is very difficult to consider efficiently new constraints in these algorithms contrary to constraint programming where search algorithms are individually done for each constraint before to be combined.

In [22] the mapping problem is tackled with genetic algorithm. As simulated annealing, genetic algorithms suffers from the sensitivity to variations of parameters, *i.e.*, the population size, the number of crossing-over points, the selection criterion, the termination criterion and the mutation probability. Moreover, as branch-and-bound the efficiency of the method depends strongly on the crossing-over algorithm and it is difficult to adapt to new models.

In [48] a dedicated heuristic is proposed to solve a mapping problem for distributed system with allocation and resource constraints and the empty-slots method, *i.e.*, time is considered to be slotted and the duration of one slot is taken as unit of time. As for other methods the heuristic gives some good results but it is really dedicated for only one kind of problem and it is difficult to adapt it with other classes of problems.

In [4] Altenbernd and Hansson propose a new clustering algorithm. A clustering algorithm merges a number of tasks to reduce teh size of the task graph and so the allocation heuristic processes mush faster. Experimental results are very significant and clustering seems a smart way for reduce the search space for large problems. However, clustering suffers from the risk of not finding any feasible assignment, but the clustering is out line of our work.

Our approach proposes to use a recent solving method: the constraint programming (see Section 4 for more details on constraint programming). The main advantage of constraints programming is that general rules are mechanically performed during the search. Thus each constraint can be considered as independent problem. A model could be simply extend by merging these different constraints. Few works exists about mapping problem with constraint programming [49,52,53,17]. Schild and Würtz [49] assumes a off-line scheduling problem for distributed real-time systems with precedence constraint. The techniques that proved to be the most successful is special global constraint and elaborate search heuristic from Operations Research. In [52,53], Szymanek and Kuchcinski focus on the problem of tasks and messages assignment and off-line scheduling under memory constraints. In [53], a novel approach is used to decide which tasks should be assigned to the same resource. Contrary to clustering, this method does not tasks into a new task, but specifies which tasks need to be executed on the same processor. This method seems improved linear clustering when multi-resource are present. Ekelin in [17] does an important work about mapping problems with multi-constraints and multi-objective functions. This work propose different methods from constraint programming to find an allocation and a off-line scheduling for distributed systems. The fact that Ekelin searches an off-line scheduling makes the problem of mapping very different with our work. The main problem with in-line scheduling is that it is not possible to simply express the schedulability test as an efficient constraint for constraint programming solver. For an off-line scheduling, the cycle window of scheduling is viewed as a discrete interval and could be naturally express by some classical constraints.

4 A short introduction to constraint programming

Constraint programming (cp) techniques have been widely used to solve a large range of combinatorial problems. They have been proved quite effective in a wide range of applications (from planning and scheduling to finance – portfolio optimization – through biology) thanks to their main advantages: declarativity (the variables, domains, constraints description), genericity (it is not a problem dependent technique) and adaptability (to unexpected side constraints).

A *constraint satisfaction problem* (csp) consists of a set V of variables defined by a corresponding set D 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 to each variable in V such that all constraints are satisfied. For example, consider 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 $C_2 : x_1 = x_3$. 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 searching systematically through the possible assignments of values to variables. The variables are labelled sequentially 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 values available.

However, cp offers more accurate methods to solve a csp. One of them is based on removing inconsistent values from variables' domains till a solution is got. 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 variables' domains, 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 deeper introduction to cp, we refer to [6].

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. Explanations consist of a set of constraints, and record enough information to justify any decision of the solver such as a reduction of domain or a contradiction. Dynamic addition/retraction of constraints are possible when explanations are maintained [?]. For example, the addition of a constraint at a leaf of the search-tree will not lead to a standard backtracking from that leaf (which could be very inefficient, as a wrong choice may have existed at the beginning of the search because the constraint was not known at that time). Instead, the solver will jump to a node appearing in the explanation and which is therefore responsible for the contradiction raised by the new constraint. More complex and more efficient techniques exist to perform intelligent repair of the solution after the addition or retraction of a constraint [28].

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 classical constraints of cp. Their translation into the csp are given under 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 lcm(T) be the least common multiple

of periods of the tasks[5]. The constraint can be written as follows:

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

- **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:

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

- **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[6]:** (*cf.* Eq. (7)) *AllDifferent*$(x_i | \tau_i \in \gamma)$

However, because of the schedulability analysis they require, timing constraints can not be translated as directly as the previous ones. Two approaches have been studied to take into account timing constraints. For the first approach, timing constraints are taken into account at each variable assignment the solver makes. It requires to conduct partial schedulability analyses and to transform their results 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 one 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 the search, a classical approach is to integrate it as a constraint of the cp model. For this purpose, some notations used previously for an allocation are extended to a partial one:

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

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

[5] 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.

[6] An *AllDifferent* constraint on a set V of variables ensures that all variables among V are different.

Definition 6.2 *We call a decision from a partial allocation $a : \mathcal{U} \to \mathcal{P}$, 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 $a' : \mathcal{U} \bigcup \Delta \to \mathcal{P}$ 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 are considered. $hp_i(a)$, respectively $lp_i(a)$ is the set of higher, respectively lower, priority tasks than τ_i on the same processor than $a(\tau_i)$.

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 which may remove some values that are inconsistent with C; and that does not remove any consistent values [45]— . 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 allocation (line 1). It is easily done by computing worst-case response time of only allocated tasks and present messages (property 11.1, Appendix 1). If the allocation found is schedulable the filtering algorithm removes values that are inconsistent (lines from 2 to 13) by testing the schedulability for each value of each remaining variable (line 7). Then, the pruning is propagated within the constraint network until a fix point is reached (while loop).

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 task $\tau_i$ not allocated do
 5:       for each $p$ in $\tau_i$'s domain do
 6:          $a'' := a' + \delta_1$ with $\delta_1(\tau_i) := p$
 7:          if ! checkSchedulability($a''$) then
 8:             Remove $p$ from domain of $\tau_i$ {Propagate to other constraints}
 9:             flag := true
10:          end if
11:       end for
12:    end for
13: end while
```

6.1 Filtering algorithm

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 (for the subproblem considered within the constraint scope and not the whole initial problem) is eliminated. This involves providing polynomial necessary and sufficient conditions regarding the existence of a solution for the constraint (according to current variables' domains).

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

This can be easily understood by reducing the problem to a graph coloring problem. Let $H = (E_1, \ldots, E_c)$ be a hyper-graph (a family of parts of $\mathcal{T}$ such that $E_i \subsetneq E_j$) where each node is mapped to a task and each hyper-edge E_i corresponds to a set of incompatible tasks according to schedulability. A necessary and sufficient condition of schedulability is therefore to know whether there exists a m-coloring of H (the color denotes the processor assigned to each node) which is a known NP-Complete problem. It is therefore not possible to compute in advance all incompatible subsets of tasks.

Our filtering algorithm is therefore based on a relaxation of the GAC. Schedulability is checked for allocated tasks and messages only, and each value of the domain of non allocated tasks (their possible processors) is checked by schedulability analysis until no more tasks can be allocated. Indeed, pruning some values of a task domain may instantiate it to a processor, leading this processor to be impossible for other tasks and so on... This is a simple hypothetical reasoning which can be understood as a singleton consistency from a constraint point of view. This filtering can however miss powerful deductions. Consider a simple example of five tasks τ_i with $\{p_1, p_2\}$ as domain such that every triple (10 triples exist) are unschedulable together (Table 6.1 gives an example). Assigning five tasks to two processors will force to place three of them together which will raise a contradiction as any triple is forbidden. However, such 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). This approach is moreover already very costly and we will now discuss the ways to speed it up.

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
Tasks and messages characteristics

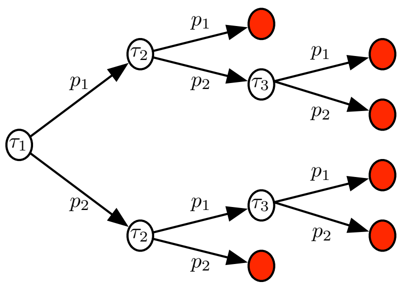

Fig. 3. Tree-search for an allocation with the global constraint. Colored nodes are nodes where the problem is inconsistent.

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 allocation. This incrementality could be used two times in the filtering algorithm. Firstly, during the schedulability test and, secondly, during the propagation by reducing the number of values to check into variables' domains.

6.2.1 Incrementality of the schedulability test

Consider a partial but schedulable —from now, we suppose that all other constraints are met, and so the partial allocation is valid too— 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 —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 on 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, for each processor p_k, schedulability has to be checked only for the tasks $\tau_i \in \Delta$ such that $a'(\tau_i) = p_k$ and the tasks $\tau_j \in \mathcal{U}$ such that $a'(\tau_j) = p_k \wedge prio_i \geq prio_j$.

The same reasoning can be applied for message schedulability analysis.

Rule 2 *When a task allocation from a schedulable partial allocation a gives rise to one message instantiation 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 $C_{ab} > \max_{M' \in lp_{ij}(a)}\{C'\}$.

The Rule 2 can be extended to a decision which 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 function CheckSchedulability(a') called at lines 1 and 7 in algorithm 1.

6.2.2 Incrementality of variable's domain

By considering the previous partial allocation and the current decision, the number of values in a variable's domain that have to be checked to remove inconsistency values (line 5 in the 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 a processor p_k 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 can't conclude on the schedulability of the system without checking it.

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

Remark, for the implementation of this rule, processors where a task is allocated during the propagation, must be take into account in turn.

6.3 Reducing further schedulability tests

Other rules can be introduced to reduce variables domain's during propagation by considering some dominance relationship 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 to be unschedulable, then p_k has to be removed from the domain of all tasks τ_j with $prio_j > prio_b \wedge C_j \geq C_i \wedge T_j \leq T_i$.*

At last, 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 which exchanges data with τ_i.*

These rules can be easily implemented in the algorithm 1 at the line 8.

7 Solving the problem with logic-based Benders decomposition

Oppositely to the global constraint strategy that includes the schedulability into 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 deeper description refer to [13]. Our approach is based on an extension of a Benders scheme. A Benders decomposition [8] 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 infeasibility 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 which is learnt by the master. We refer here to a more general Benders scheme called *logic Benders decomposition* [25] where any kind of subproblems can be used as long as the inference dual of the subproblem can be solved.

We propose an approach inspired from methods used to integrate constraint programming into a logic-based Benders decomposition [55,9]. 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 which are unschedulable for the same reason.

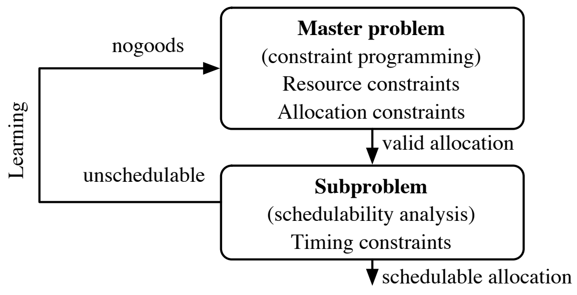

Fig. 4. Logic-based Benders decomposition to solve an allocation problem

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 (as 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.

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 sure 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*[7] 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.

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

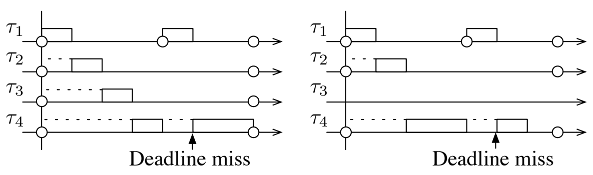

Fig. 5. Illustration of a schedulability analysis. The task τ_4 does not meet its deadline. The subset $\{\tau_1, \tau_2, \tau_4\}$ is identified to explain the unschedulability of the system.

But 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 of the order of enumeration of $hp_i(A)$. In the example, $\{\tau_1, \tau_t 3, \tau_4\}$ is also a minimal set. However, the more constraints are, more slow the solver is, that why 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* [27] (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:

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

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 term of constraint yields to:

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

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

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

Algorithm 2 Minimal task set

```
QuickXPlainTask($\tau_i, A, D_i$)
 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 $X \cup \{\tau_i\}$
```

7.3 Applying the method to an example

An example to illustrate the theory is developed hereafter. It will show how the cooperation between master- and sub-problems 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 $prio_{ij} = y$.

p_i	p_0	p_1	p_2	p_3
m_i	102001	280295	360241	41617

Table 2
Processor characteristics

The problem is constrained by:

- residence constraints:
  - 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 constraint:
  - 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 here out of our purpose. The valid solution it returns is:

τ_i	T_i	C_i	μ_i	$prio_i$	Message
τ_0	36000	2190	21243	1	600,1 $\rightarrow$ 13
τ_1	2000	563	5855	6	500,3 $\rightarrow$ 8
τ_2	3000	207	2152	15	600,7 $\rightarrow$ 7
τ_3	8000	2187	21213	3	
τ_4	72000	17690	168055	7	300,4 $\rightarrow$ 9
τ_5	4000	667	6670	8	800,5 $\rightarrow$ 19
τ_6	12000	3662	36253	14	
τ_7	3000	269	2743	16	
τ_8	2000	231	2263	12	100,6 $\rightarrow$ 18
τ_9	72000	6161	59761	9	
τ_{10}	12000	846	8206	4	200,2 $\rightarrow$ 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 $\rightarrow$ 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

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

First step: analysing 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: analysing 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 minimal set. Consequently, the explanation of the unschedulability is translated into the new constraint:

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

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

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

for τ_{16}:

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

and for τ_{19}:

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

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

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

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 [14] that implements our solving approaches. The method with global constraint as Global-CPRTA and theBenders decomposition method is denoted by Benders-CPRTA. It is based on the CHOCO [30,?] cp system and PALM [28], an explanation-based cp system.

For the allocation problem, no specific benchmarks are available as a point of reference in the real-time community. Experiments are usually done on didactic examples [56,?] or randomly generated configurations [43,?]. We opted for this last solution. Our generator takes several parameters into account:

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

Task periods and priorities are randomly generated. Worst-case execution times are initially randomly chosen and evaluated again so as: $\sum_{i=1}^{n} C_i/T_i = m\%_{global}$. The memory need of a task is proportional to its worst-case execution time. Memory capacities are randomly generated while satisfying: $\sum_{k=1}^{m} m_k = (1 + \%_{mem}) \sum_{i=1}^{n} \mu_i$. For a sake of simplicity, only linear data communications between tasks are considered and the priority of an edge is inherited from the task producing it.

The number of tasks involved in allocation constraints is given by the parameters $\%_{res}$, $\%_{co}$, $\%_{exc}$. Tasks are randomly chosen and their number (involved in co-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 utilization 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 ($\%_{res}$, $\%_{co}$, $\%_{exc}$). 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 presence of data exchanges impacts on both problems and the difficulty has been characterized by the ratios mes/n and $\%_m$. $\%_m$ expresses the impact of data exchanges on schedulability analysis by linking periods and message sizes.

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/n	$\%_m$
1	60	1	0	0	0	1	40	1	0	0
2	30	2	15	15	15	2	60	2	0.5	70
3	10	3	33	33	33	3	90	3	0.875	150

Table 4
Details on difficulty classes

8.1 Results

Table 5 summarizes some of the results of experiments with Benders-CPRTA and Global-CPRTA. 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 does not exhibit a specific behaviour. $\%_{\mathrm{RES}}$ gives the number of problem instances successfully solved (a schedulable solution has been found or it has been proved that none exists) within the time limit of 10 minutes per instance. $\%_{\mathrm{VAL}}$ gives the percentage of schedulable solutions found (thus $\%_{\mathrm{RES}} - \%_{\mathrm{VAL}}$ gives the percentage of inconsistent problems). CPU is the mean computation time in seconds. ITER is the number of iterations between the master problem and the subproblem for the Benders-CPRTA. NOG is the number of nogoods inferred from the subproblem for the Benders-CPRTA. The data are obtained in average (on instances solved within the required time) on 100 instances (40 tasks, 7 processors) per class of difficulty with a Pentium 4 (3.2 GHz).

First, by examining the CPU column, we notice that Benders- and Global-CPRTA still remain very efficient in spite of their seeming complexity. Moreover, for Benders-CPRTA as measured by ITER and NOG, the cooperation between master- and sub-problems is quite significant and the learning is of some importance.

The lines 1 to 5 in Table 5 show results for high difficulty classes without communications between tasks. The results in lines 1 to 3 are very good. They illustrate the basic ability of cp to consider memory and allocation constraints. Lines 4 and 5 display some performances that are going down when the schedulability difficulty increases.

Indeed, 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, this approach integrates schedulability analysis into the cp algorithm so as not "to delay" its taking into account, the line 5 shows that for consistent problems. However, Benders-CPRTA is an average better than Global-CPRTA.

The lines 6 to 8 deal with allocation problems where tasks may communicate. Once more, one can notice that when data exchanges increase (and thus message exchanges on the bus too), the CPRTA performances decrease. Reasons are the same as those of task schedulability: the more the messages are on the bus, the more their scheduling becomes difficult. Moreover, we have observed that nogoods inferred from message unschedulability are usually "weaker" (the search space cut is smaller) than the ones inferred from task unschedulability. Learning is then less efficient for this kind of problems.

8.2 Comparison with simulated annealing

As to get comparative performances for CPRTA, a simulated annealing (SA) algorithm, inspired from [56], has been implemented. In [56] 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 too. The implementation has been optimized so as to reduce computation time of this energy function.

SA is a heuristic method. As a consequence, in our case, it can only conclude on problems with a solution. In Table 6 only CPRTA results for such problems are compared to SA. As seen on Table 6, except for problems for which CPRTA

		Benders-CPRTA					Global-CPRTA		
	cat.	$\%_{\mathrm{RES}}$	$\%_{\mathrm{VAL}}$	CPU	ITER	NOG	$\%_{\mathrm{RES}}$	$\%_{\mathrm{VAL}}$	CPU
1	2-2-2-1	100.0	56.0	0.6	13.5	95.2	91.0	56.0	25.5
2	3-2-2-1	98.0	56.0	2.6	31.0	133.2	85.0	50.0	43.5
3	2-3-2-1	100.0	19.0	0.7	7.6	43.5	89.0	19.0	31.3
4	1-1-3-1	88.0	88.0	29.2	95.7	471.6	86.0	86.0	32.3
5	2-2-3-1	72.0	13.0	28.8	13.1	59.7	71.0	24.0	48.6
6	2-2-2-2	100.0	70.0	1.5	19.5	69.9	91.0	67.0	12.3
7	1-2-2-3	78.0	55.0	70.5	296.3	60.7	50.0	31.0	74.9
8	2-2-2-3	68.0	48.0	52.1	148.7	117.2	42.0	27.0	79.1

Table 5
Average results on 100 instances randomly generated into classes of problems

must be improved (see Section 8.1), CPRTA produces as satisfactory results as SA does. Moreover, it should be pointed out that even if CPRTA is sometimes less efficient than SA, CPRTA solves on average more problems than SA does if we take into account problems without solution.

	SA		Benders		Global	
cat.	$\%_{\mathrm{VAL}}$	CPU	$\%_{\mathrm{VAL}}$	CPU	$\%_{\mathrm{VAL}}$	CPU
2-2-2-1	56.0	2.8	56.0	1.0	56.0	8.1
3-2-2-1	58.0	22.3	56.0	4.3	50.0	35.5
2-3-2-1	17.0	35.2	19.0	2.6	19.0	4.0
1-1-3-1	99.0	3.1	88.0	29.2	86.0	32.3
2-2-3-1	20.0	76.4	13.0	157.7	24.0	41.5
2-2-2-2	68.0	14.7	70.0	1.6	67.0	47.3
1-2-2-3	64.0	63.7	55.0	100.4	31.0	99.5
2-2-2-3	63.0	44.1	48.0	73.5	27.0	86.0

Table 6
Comparison between CPRTA and SA

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 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 in addition [28].

Thus, in the case of an allocation problem for which no solution has been found, we analyse 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. We will first focus on the characteristics of the software architecture by analysing how each task is "responsible" for the failure. We will give there some insight on the way a critical task from the schedulability point of view can be identified. Each failure of the search process due to schedulability is analysed and transformed into a constraint criterion that encapsulates an accurate reason for this failure. The study of those criteria may lead to the guilty tasks. 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 stronger 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:

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

This criterion considers the presence of a task in each constraint and its impact. Bigger $\mathcal{C}_i$ is, 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 an other period) and so to obtain a solution for the problem.

Table 7 gives $\mathcal{C}_i$ obtained on the example of the Section 7.3 with ŒDIPE [14].

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 CPRTA, we found a solution for this problem.

Notice that this process consists in analysing the final set of constraints with a heuristic based on the information gathered during the search. This process can be generalized to memory and allocation constraints by the use of a specific search technique [44] even if explicit reasons for failure on memory or allocation are not kept in memory in our current approach (contrary to schedulability one).

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 7
Constraint criterions computed on example

10 Conclusion and future work

In this paper, we present an original and complete approach (CPRTA) to solve a hard real-time allocation problem. To tackle this problem, two approaches are proposed. For the first one, timing constraints have conducted to define a global constraint. This method has been optimised to speed-up the search by exploiting specific properties of schedulability analysis. For the second one, a decomposition method which is 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 sub-problems 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 constraints.

Experimental results show that these two methods produce an efficient way to solve allocation problem. Thanks to the learning that produces significant knowledge about schedulability, the CPRTA based on Benders decomposition achieves better performance.

An other 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 works, our aim is to integrate into the design process an intelligent tool based on CPRTA ables to return pertinent explanations justifying the failure.

11 Acknowledge

The authors wish to acknowledge ... for the proof read that have helped to improve this paper.

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 *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002)*, volume 29, June,.

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

[6] R. Barták. Constraint programming: In pursuit of the holy grail. In *Proc. of the Week of Doctoral Students (WDS99)*, 1999.

[7] J.-P. Beauvais. *Etude d'Algotithmes de Placement de Tches Temps Rel Priodiques Complexes dans un Systme Rparti.* PhD thesis, Ecole Centrale de Nantes, 1996.

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

[9] 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.

[10] G. Borriello and D. Miles. Task scheduling for real-time multiprocessor simulations. In *Proc. of 11th Workshop on RTOSS*, pages 70–73, 1994.

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

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

[13] 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.

[14] H. Cambazard and P.E. Hladik. ŒDIPE.

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

[16] 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.

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

[18] 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.

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

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

[21] 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.

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

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

[24] 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.

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

[26] 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.

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

[28] N. Jussien. The versatility of using explanations within constraint programming. Technical Report RR 03-04-INFO, École des Mines de Nantes, 2003.

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

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

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

[32] 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.

[33] 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.

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

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

[36] 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.

[37] 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.

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

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

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

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

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

[47] F. E. 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.

[48] 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.

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

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

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

[52] 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.

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

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

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

[56] 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.

[57] K. W. Tindell, H. Hansson, and A. J. 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.

[58] L. Vargas and R. Olivaira. Empirical study of tabu search, simulated annealing and multi-start in fieldbus scheduling. In *Proc. of the 10th IEEE ETFA*, volume 1, pages 101–108, 2005.

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

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

[61] J. Yeckle and W. Rivera. Mapping and characterization of applications in heterogeneuous distributed systems. In *Proc. of the 7th World Multiconference on Systemics, cybernetics and informatics (SCI2003)*, 2003.

Appendix 1: schedulability analysis properties

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,

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

The same reasoning could be done 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 allocation a and two other ones 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. $R_i^{(0)}(a') = C_i + \sum_{\tau \in hp_i(a')} C \leqslant C_j + \sum_{\tau \in hp_j(a'')} C = R_j^{(0)}(a'')$.

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

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

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

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 other ones 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,

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

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