

# How to Solve Allocation Problems with Constraint Programming

Pierre-Emmanuel Hladik<sup>1</sup>, Hadrien Cambazard<sup>2</sup>, Anne-Marie Déplanche<sup>1</sup>, Narendra Jussien<sup>2</sup>

<sup>1</sup> IRCCyN, UMR CNRS 6597

1 rue de la Noë – BP 9210  
44321 Nantes Cedex 3, France

{hladik,deplanche}@irccyn.ec-nantes.fr

<sup>2</sup> École des Mines de Nantes, LINA CNRS

4 rue Alfred Kastler – BP 20722  
44307 Nantes Cedex 3, France

{hcambaza,jussien}@emn.fr

## Abstract

*In this paper<sup>1</sup>, we present a constraint programming-based approach to solve a hard real-time allocation problem. This problem consists in assigning periodic tasks to processors in the context of fixed priority preemptive scheduling. Our approach builds on dynamic constraint programming together with a learning method to find a feasible processor allocation under constraints. This problem is decomposed into two subproblems: allocation, and schedulability. Benders decomposition is then used as a way of learning when the allocation subproblem yields a valid solution while the schedulability analysis of the allocation does not. The rationale of this approach is to learn from the failures of the schedulability analysis to reduce the search space.*

## 1. Introduction

Real-time systems have applications in many industrial areas: telecommunication systems, automotive, aircraft, robotics, etc. Today applications (e.g., cars) involve many processors in order to be able to serve different purpose demands (e.g., cruise control, ABS, engine management, etc.).

The problem consists in assigning periodic tasks to processors in the context of fixed priorities preemptive scheduling. We assume that the characteristics of the tasks (execution time, priority, etc.) and the physical architecture (processors and networks) are known.

The problem of assigning a set of hard preemptive real-time tasks in a distributed system belongs to a class of NP-Hard combinatorial problems. It has been tackled with *ad-hoc* approaches, simulated annealing and genetic algorithms. Recently, Ekelin [5] have used constraint programming to produce an assignment and a pre-runtime scheduling of distributed systems under optimization criteria.

This paper presents a decomposition-based method (related to logic Benders-based decomposition [6]) which sep-

arates the allocation problem from the scheduling one: the allocation problem is solved by means of constraint programming tools, whereas the scheduling problem is treated with traditional real-time schedulability analysis. The main idea is to "learn" from the schedulability analysis failures to re-model the allocation problem and reduce the search space. In that sense, we can compare our approach to a form of learning from mistakes.

The remainder of this paper is organized as follows. In Section 2, we describe the problem. Section 3 is dedicated to the description of the master/subproblems and the cooperation between them. The logical Benders decomposition scheme is briefly introduced and the links with the approach are put forward. An experimental case study is presented in Section 4.

## 2 Problem description

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

The hardware architecture consists of a set  $\mathcal{P} = \{p_1, \dots, p_k, \dots, p_m\}$  of  $m$  identical processors with fixed memory capacity  $m_k$  and identical processing speed. They are all connected to a network with bandwidth  $\delta$  and with token ring protocol. This protocol was chosen by similarity with Tindell's approach [10]. At this time, an extension to CAN protocol are proceeded.

To model the software architecture, we consider a valued, oriented and acyclic graph  $(\mathcal{T}, \mathcal{C})$ . The set of nodes  $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$  corresponds to the tasks and the set of edges  $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$  refers to the messages sent between tasks.

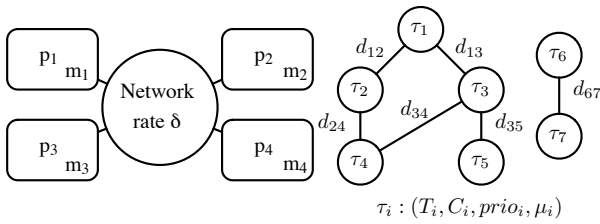
A task  $\tau_i$  is defined through its temporal characteristics and resource needs: its period  $T_i$  (as a task is periodically activated), its worst-case execution time without preemption  $C_i$  and its memory need  $\mu_i^2$ . Edges  $c_{ij} = (\tau_i, \tau_j) \in \mathcal{C}$

<sup>1</sup>The authors wish to acknowledge Frédéric Desobry for the constructive comments that have helped to improve this paper.

<sup>2</sup>Within this model are only considered static real-time systems in which all memory resources are known and allowed.

are valued with the amount of exchanged data:  $d_{ij}$ . Communicating tasks have the same activation period. Moreover, they are able to communicate in two ways: a local communication without any delay which uses the memory of the processor and requires the tasks to be located on the same processor, and a distant communication which uses the network. In both situations, we do not consider any precedence constraints. Tasks are periodically activated in an independent way, and they read and write data at the beginning and the end of their execution.

Finally, each processor is scheduled with a fixed priority strategy. A priority  $prio_i$  is given to each task. Task  $\tau_j$  has priority over  $\tau_i$  if and only if  $prio_j < prio_i$ . A task execution may be preempted by tasks with higher priority.



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

An allocation is a mapping  $A : \mathcal{T} \rightarrow \mathcal{P}$  such that the image of a task  $\tau_i$  is a processor  $p_k$ :

$$\tau_i \mapsto A(\tau_i) = p_k \quad (1)$$

The allocation problem consists in finding the mapping  $A$  which respects the whole set of constraints described in the immediate below.

There are three classes of constraints the allocation problem must respect: timing, resource, and allocation constraints.

- **Allocation constraints:** This set of constraints deal with the position (or relative position) of the tasks on the processors. Some tasks require specific processor characteristics to be executed (signal processor, compression processors, databases, etc.) and can only reside on a subset of the available processors. Others must not be put together on the same processor. Two sets of tasks may also have to be disjoint in any assignment.
- **Resource constraints:** The memory usage of a processor cannot exceed a fixed capacity.
- **Timing constraints:** A hard real-time system must respect all timing constraints to assure the security of the

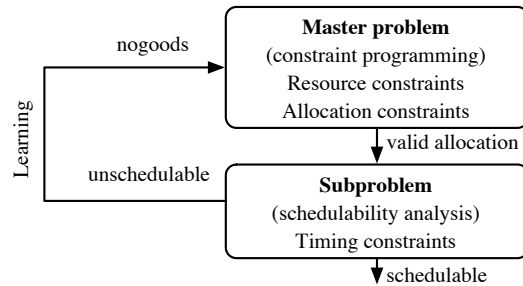
process. Temporal constraints define a schedulable allocation according to deadlines or due dates requirements.

An allocation 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 Solving the problem

Constraint programming (CP) techniques have been widely used to solve a large range of combinatorial problems. 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. This mechanism coupled with a backtracking scheme allows the search space to be explored in a *complete way*. For a deeper introduction on CP, we refer to [1].

We propose an approach inspired from methods used to integrate constraint programming into a logic-based Benders decomposition [2, 6]. The allocation and resource problem are considered on one side, and schedulability on the other (see Fig. 2). The master problem solved with constraint programming yields a valid allocation. The subproblem checks the schedulability of this allocation, 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.



**Figure 2. Solving an allocation problem through logic-based Benders decomposition.**

#### 3.1 The master problem

As the master problem is solved using constraint programming techniques, we need first to translate our problem into CSP. The model is based on a redundant formulation using three kinds of variables:  $x, y, w$ .

Let us first consider  $n$  integer-valued variables  $x$  which are decision variables and correspond to each task, representing the processor selected to process the task:  $\forall i \in$

$\{1..n\}$ ,  $x_i \in \{1, \dots, m\}$ . Then, boolean variables  $y$  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  $w$  are introduced to express whether a pair of tasks exchanging a message 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.

One of the main objectives of the master problem is to solve efficiently the assignment part. It handles two kinds of constraints: allocation and resources. Due to the lack of space, we do not recall the close-form expressions for these constraints and refer to [4] for any further detail.

### 3.2 Subproblem(s)

The subproblem we consider here is to check whether a valid solution produced by the master problem is schedulable or not. Deadlines can correspond to non-communicating tasks if no data is sent or if the receiver is on the same processor, and to communicating tasks when the receiver is allocated on another processor. A non-communicating task deadline equals the task period:  $D_i = T_i$ ; communicating task deadlines equal the task period minus the maximum amount of time needed for transmitting a message:  $D_i = T_i - TRT$  (this ensures a regular data refreshment).

A task is schedulable if its worst-case response time is lower than its deadline.

Classical calculation of worst-case response times and evaluation of the maximum transmission delay on a token ring network are given in the immediate following.

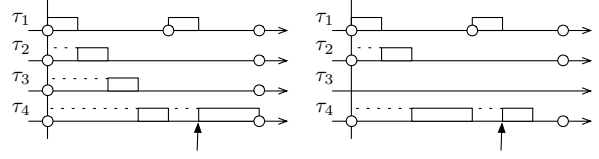
**Worst-case response time.** For independent tasks, it has been proved that the worst execution scenario for a task  $\tau_i$  happens when it is released simultaneously with all the tasks which have a priority higher than  $\tau_i$ . The worst-case response time for  $\tau_i$  writes:

$$R_i = C_i + \sum_{\tau_j \in hp_i(A)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (2)$$

with  $hp_i(A)$  the set of tasks with a priority higher than  $\tau_i$  and located on the processor  $A(\tau_i)$  for a given allocation  $A$ . Worst-case response time  $R_i$  is then easily obtained by looking for the fix-point of Eq. (2).

**Transmission time on a token ring.** The maximum transmission delay on a token ring network is bounded: let  $TRT$  (Token Rotation Time) denote the maximum duration for sending data on the network. An upper bound on this duration was proposed by Tindell in [10] and is computed by taking into account all the messages to be sent on the network:

$$TRT = \sum_{\substack{c_{ij} = (\tau_i, \tau_j) \\ A(\tau_i) \neq A(\tau_j)}} \frac{d_{ij}}{\delta} \quad (3)$$



**Figure 3. The task  $\tau_4$  does not meet its deadline. The subset  $\{\tau_1, \tau_2, \tau_4\}$  is identified to explain the unschedulability of the system.**

### 3.3 Cooperation between master and subproblem(s)

We now consider a valid allocation in which some tasks are not schedulable. Our purpose is to explain why this allocation is unschedulable, and translate this into a new constraint for the master problem. For our problem, we separate two kinds of explanations.

**Non-communicating tasks.** The explanation for the unschedulability of a non-communicating task  $\tau_i$  is the presence of tasks with higher priority on the same processor (let  $hp_i(A)$  denote the set of tasks with higher priority). If another allocation with  $\tau_i$  and  $hp_i(A)$  is on the same processor, it is sure that  $\tau_i$  will still be unschedulable. So the master problem must be constrained so that all solutions where  $\tau_i$  and  $hp_i(A)$  are together are not considered any further. This constraint corresponds to a *NotAllEqual*<sup>3</sup> on  $x$ :

$$NotAllEqual(x_i | \tau_i \in hp_i(A) \cup \{\tau_i\})$$

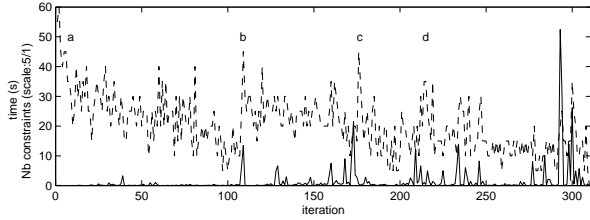
It is worth noticing that this constraint could be expressed as a linear combination of variables  $y$ . However, *NotAllEqual*( $x_1, x_3, x_4$ ) excludes the solutions that contain the tasks  $\tau_1, \tau_3, \tau_4$  gathered on *any* processor.

However, we can see that this constraint is not relevant. For example, in Fig. 3, the set  $\{\tau_1, \tau_2, \tau_3, \tau_4\}$  is unschedulable. It explains the unschedulability but is not minimal in the sense that if we do not consider one task, the set is still unschedulable. Here, the set  $\{\tau_1, \tau_2, \tau_4\}$  is sufficient to justify the unschedulability.

In order to derive more precise explanations (*i.e.*, achieve a more relevant learning), a conflict detection algorithm, namely *QuickXplain* [7], has been used to determine a minimal (*w.r.t.* inclusion) set of involved tasks.

**Communicating tasks.** The difficulty here is to avoid incriminating the whole system. If a communicating task  $\tau_i$  is unschedulable, it is because of the tasks in  $hp_i(A)$  and the

<sup>3</sup>A *NotAllEqual* on a set  $V$  of variables ensures that at least two variables among  $V$  take distinct values.



**Figure 4. Execution of a hard instance. Resolution time (line, in seconds) and number of learned constraints (dotted line, with a scale of 5/1) inferred at each iteration. (310 iterations, 1279 constraints).**

message set  $M(A)$  transmitted on the network. The translation of this information in terms of constraints yields:

$$\text{NotAllEqual}(x_i | \tau_i \in hp_i(A) \cup \{\tau_i\}) \vee \sum_{c_{ij} \in M} w_{ij} < \#M(A)$$

We have:  $w_{ij} = 1$  if  $c_{ij}$  is transmitted on the network, so if a message of  $M(A)$  is not transmitted, the sum becomes lower than  $\#M(A)$  and the constraint is satisfied.

Again, we have used QUICKXPLAIN to refine the information, it is arranged to take into account message and task.

#### 4 An experimental case study

We chose to implement our approach with a tool named ŒDIPE [3], which is based on the CHOCO [9] constraint programming system and PALM [8], an explanation-based constraint programming system.

We conducted different experimentations [4], but due to the lack of space we only present here an experimental case study.

The execution of a particular instance of independent tasks (40 tasks, 7 processors, 60% global utilization factor of processors, 30% memory over-capacity, *i.e.* the amount of additional memory available on processors with respect to the needs in memory of all tasks, 5 residence constraints, 5 co-residence constraints and 5 exclusion constraints) is outlined on Fig. 4. Resolution time and learned constraints at each iteration between master and subproblems are plotted.

The master problem adapts the current solution to the nogoods thanks to its dynamic abilities and the learning process is very quick from the beginning. The number of learned constraints decreases until a hard satisfaction problem is formulated (*a-b* in Fig. 4). The master problem then is forced to evaluate an important amount of choices to provide a valid allocation (*b*). The process starts again with a quick learning of nogoods (*b-c*, *c-d*). This example shows the efficiency of using a learning method to guide the search for solution.

#### 5 Conclusion and future work

In this short paper, we presented an original approach to solve the allocation problem with constraint programming. We use a decomposition method which is, up to a certain extent, built on a logic Benders decomposition for learning. The overall problem is split into a master problem for allocation and resource constraints and a subproblem for timing constraints, using the learning technique in an effort to combine the various issues into a solution that satisfies all constraints.

First experiments have encouraged us to go a step further. Our approach offers a new answer to the problem of real-time task allocation. It opens new perspectives on integrating techniques coming from a broader horizon, other than optimization, within CP in a Benders scheme.

Short-term perspectives for this on-going work include the implementation of different heuristics to assist and speed up the search. Future work includes (but is not limited to) the extensive comparison of our approach with other methods such as traditional constraint and linear programming. We also believe that the extension of this work to other kinds of network protocols such as CAN, TDMA, etc, and precedence constraints is of serious interest.

#### References

- [1] R. Barták. Constraint programming: In pursuit of the holy grail. In *Proc. of WDS99*, 1999.
- [2] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
- [3] H. Cambazard and P. Hladik. ŒDIPE. [www.irccyn.ec-nantes.fr/irccyn/d/fr/equipements/TempReel/logs/software-9-oedipe](http://www.irccyn.ec-nantes.fr/irccyn/d/fr/equipements/TempReel/logs/software-9-oedipe).
- [4] 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 *Proc. of CP 2004*, 2004.
- [5] C. Ekelin. *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. PhD thesis, Chalmers University of Technology, 2004.
- [6] J. N. Hooker and G. Ottoson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
- [7] U. Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *Proc. of IJCAI 01*, 2001.
- [8] N. Jussien. The versatility of using explanations within constraint programming. Technical Report RR 03-04-INFO, École des Mines de Nantes, 2003.
- [9] F. Laburthe. Choco: implementing a cp kernel. In *proceedings of CP 00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems*, 2000.
- [10] 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.