Real-time scheduling for systems with precedence, periodicity and latency constraints

Liliana Cucu[1], *Rémy Kocik*[2], *Yves Sorel*[1]

(1) INRIA Rocquencourt,
BP 105 - 78153 Le Chesnay Cedex, France
liliana.cucu@inria.fr, yves.sorel@inria.fr

(2) ESIEE,
Cité Descartes - BP 99 - 2, Bd Blaise Pascal
93162 Noisy-le-Grand Cedex, France
kocikr@esiee.fr

Abstract

First we present the main results concerning, in the one hand systems with periodicity constraints and deadlines, and in the other hand systems with precedence constraints and deadlines, in both cases for one computing resource. Then, we give a model in order to state clearly the problem for scheduling systems with precedence, periodicity and latency constraints. In order to solve this problem we give a nonpreemptive, off-line scheduling algorithm which uses in turn an algorithm of latency marking. We demonstrate the optimality of the scheduling algorithm, and after proving the equivalence of the notions of latency constraint and deadline, we extend this latter algorithm for scheduling real-time systems with precedence, periodicity constraints and deadlines for one computing resource.

Keywords: algorithm, scheduling, optimality, real-time, periodicity, latency, precedence, deadline.

1 Introduction

In scheduling theory of real-time systems, we have two main interests: periodic systems and systems with precedence constraints. Even if both fields are separately rich in results in the literature, to our best knowledge there are few results where both aspects are treated together. The purpose of this paper is to propose, in the one hand a model which deals with both aspects, and in the other hand to solve the problems of finding a schedule (if there is one) which satisfies the periodicity, latency and the precedence constraints of the system, in the case of one processor. The paper starts with notations used to present the main results for periodic systems and for systems with precedence constraints. The next section presents the model used to describe the problem to be solved. The next section presents the algorithm for this problem with the demonstration of its optimality. The paper ends with a conclusion and further research.

2 Notations and results

In order to clearly distinguish the specification level and its associated model we are mainly interested in, from the implementation level, we use the term *operation* instead of the commonly used "task" too closely related to the implementation level. For the same reason we use the term *operator* instead of "processor" or "machine".

The results often found in the literature are given according to three parameters: $[\alpha/\beta/\gamma]$ [1], where α gives the number of operators and specifies if they are of the same type (homogeneous machine) or not, β the operation characteristics (period, deadline, computation time, precedence constraints) and γ the optimality criterion (a relation that allows us to compare possible solutions of the problem). For an operation, we may specify a computation time C, a period T, a deadline D (defined from the start either of the period, either of the schedule), a release time r, a start time s and a jitter $J_i = |(s_{i+1} - r_{i+1}) - (s_i - r_i)|, i \in \mathbb{N}$ with $0 \leq C \leq D \leq T$ and $r \leq s$ (see figure 1).

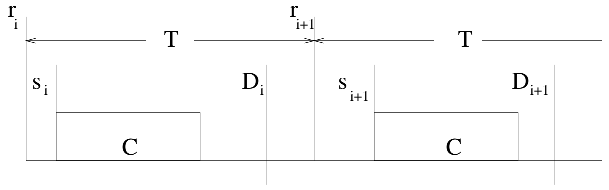

Figure 1: Basic real-time model

For the problem with periodic operations and without precedence constraint [*1/* $D = T$*/*], Liu and Layland give the RMS algorithm [2], improved by Lehoczky and Sha (introduction of "blocking") [3] and by Audsley (introduction of jitter by holistic analysis,[4]) and the EDF algorithm for [*1/* $D \leq T$*/*].

The precedence constraints are given by a partial order on the execution of the operations ($A \rightarrow B$ means that B cannot start before A is executed) that may be represented by a directed acyclic graph. The partial order associated to this graph defines a *potential parallelism* [5] on the set of operations. For example in figure 2, B and C may be executed in parallel on different operators if we are in the case $\alpha > 1$, whereas A and B must be executed sequentially.

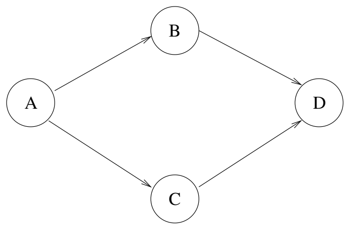

Figure 2: Precedence constraints graph

In the case of systems with precedence constraints where all the operations have the same release time [*1/ precedence/ minimize the maximum lateness*], Lawler gives the "first-to-last-rule" which is optimal [6, 7]. If the operations do not have the same release time we obtain a NP-hard problem [8], but if preemption is allowed the problem is polynomial [9]. Also, for the problem [*1/precedence, D/*] Blazewicz gives an polynomial solution [10].

In the general case of systems for which all the operations may have precedence, latency and also periodicity constraints, to our best knowledge, there is no result for the problem with one operator.

3 The model and the problem to solve

Each operation may belong to a precedence constraint, i.e. belongs to a pair defining the partial order, or/and each operation may have a periodicity constraint. Moreover, an operation may be repeated leading to several instances of the same operation either spatially without precedence constraint between consecutive repetitions, defining a local potential parallelism, or temporally with precedence constraints between consecutive repetitions. Actually, the real-time system interacts with the physical environment [11], therefore the graph of precedence constraints is a *pattern* temporally infinitely repeated [5]. If one consider only the pattern itself, according to its partial order the first operations are called inputs and the last operations are called outputs. These operations correspond to respectively sensors and actuators. For example in the pattern of figure 3, A is repeated temporally three times whereas C is repeated spatially two times, A_1 is the input operation, and C_1, C_2 are the output operations. Actually, the infinite repetition of the pattern induces an infinite repetition of all the operations.

In order to simplify this complex graph model, that we call *not factorized graph*, we define a *finite repetitive precedence constraint* when finitely repeated (spatially or temporally) operations are in relation of precedence constraint. This allows to represent in

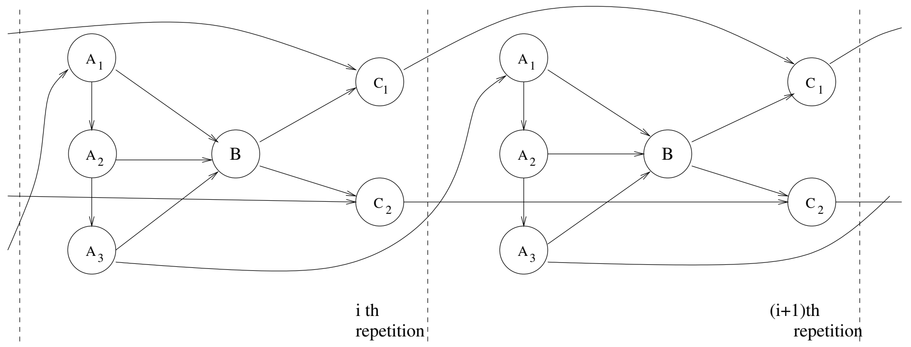

Figure 3: Not factorized graph

the model only one instance of each repeated operation with its number of repetitions, and to describe synthetically with a function *code* the existing edges between two different operations possibly belonging to different repetitions, or between the same operations belonging to different consecutive repetitions. All these edges are intra-pattern. Due to the infinite repetition of the pattern, two input operations or two output operations, belonging to two consecutive patterns, may be in relation of precedence that we call *infinite repetitive precedence constraint*. It is usually the case when we have a unique sensor for an input and/or an unique actuator for an output instead of an infinite number of sensors and/or actuators. These edges are inter-pattern. The resulting graph is said factorized.

For two operations A and B repeated temporally or spatially, we have $codeAB$: $\{1, 2, .., n\} \times \{1, 2, .., m\} \rightarrow \{0, 1\}$ such as:

$$codeAB(i,j) = \begin{cases} 1, & \text{if an edge starts from the } i^{th} \text{ repetition of } A \\ & \text{and ends on the } j^{th} \text{ repetition of } B \\ 0, & \text{otherwise} \end{cases}$$

where n is the number of repetitions of A and m for B. It is obvious to demonstrate that the number of the possible edges between all the repetitions of A and B is bounded by $n \cdot m$ (we may have at most m edges from every A).

For an operation temporally repeated n times, we have $codeAA$: $\{1, 2, .., n\} \times \{1, 2, .., n\} \rightarrow \{0, 1\}$ such as:

$$codeAA(i,j) = \begin{cases} 1, & \text{for } j = i + 1 \text{ and } i < n \\ 0, & \text{otherwise} \end{cases}$$

If there is no edge between successive repetitions of A, the function $codeAA$ is not specified. When the function $codeAA$ is specified the number of the possible edges between successive repetitions of A is $n - 1$, otherwise 0.

For an input (resp. output) operation A which is temporally infinitely repeated, and also finitely temporally repeated n times (inside the pattern), we have an edge ∞ in the factorized graph, corresponding to the infinite precedence constraint in the not factorized graph. If the $codeAA$ has been defined for A then we have an edge between the last repetition A_n and the first repetition A_1 belonging to the next pattern. If no $codeAA$ has been defined, then we have an edge between A_i and the A_i belonging to the next pattern, for all the repetitions A_i, $i \in \{1, 2, \ldots, n\}$.

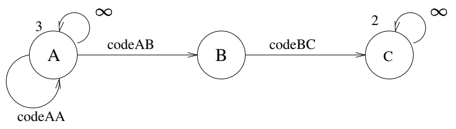

Figure 4: Factorized graph

It is obvious to demonstrate that a factorized graph contains cycles, if and only if, the not factorized graph contains cycles.

From the not factorized graph presented in figure 3 we obtain the factorized graph illustrated in figure 4 which has the following *code* functions: $codeAB$: $\{1, 2, 3\} \times \{1\} \rightarrow \{0, 1\}$ with $codeAB$(i,1)=1, $\forall i \in \{1, 2, 3\}$, $codeBC$: $\{1\} \times \{1, 2\} \rightarrow \{0, 1\}$ with $codeBC(1, i)$=1 $\forall i \in \{1, 2\}$ and the function $codeAA$ given by the general definition of the function *code* for an operation temporally repeated n times (in this case, $n = 3$). The operations A and C have an edge ∞.

Remark 3.1 *In the case where we have in the factorized graph two operations A and B with the same number n of repetitions and the function $codeAB$: $\{1, 2, .., n\} \times \{1, 2, .., n\} \rightarrow \{0, 1\}$ such that*

$$codeAB(i, j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

in order the simplify the model, the function code will not be specified for this particular case of repetitive precedence constraint. Also, when the number of repetitions of an operation is equal to 1, we do not mention it in the factorized graph.

Finally, in order to complete the model, we give the two types of the real-time constraints which we want to be satisfied by the system, namely the *latency* and the *periodicity*.

Definition 3.1 *for two different operations A and B belonging to the pattern infinitely repeated, we say that the pair (A, B) has a latency constraint L when the operations have to be scheduled such that $s_B + C_B - s_A \leq L$.*

Remark 3.2 *we must have a directed path starting with A and ending with B if (A, B) has a latency constraint, because it only concerns operations which are in relation of precedence constraint. We call A "first" in the latency constraint and B "last" in the latency constraint. We denote by $\mathcal{L}$ the set of all the pairs of operations having a latency constraint. An operation A may belong to several pairs of operations having different latency constraints.*

Definition 3.2 *for two consecutive repetitions A_i and A_{i+1} of the same operation A, we say that A has a constraint of periodicity T_A if $s_{A_{i+1}} - s_{A_i} = T_A, \forall i \in \mathbb{N}$. We denote by A_1 the first repetition of A.*

Remark 3.3 *We assume that the periodic operations are scheduled strictly with their exact periodicity constraints. Thus, it amounts to not allow any jitter. Also, we assume that all the periodicity constraints are multiples of each other.*

Without any loss of generality [12], we assume that all the operation characteristics, i.e. the fixed computation time (exactly known), the periodicity and the latency constraints, are defined as multiples of a clock tick τ (time is discrete). Afterwards the values of the given characteristics are implicitly multiplied by τ.

In the example presented in figure 3 we have $C_A = 2$, $C_B = 1$ and $C_C = 2$, and the following real-time constraints to be satisfied: the periodicity constraints $T_A = 5$ and $T_B = 15$, the latency constraints: $L(A_2, C_2) = 10$ and $L(B, C_1) = 9$. The periodicity constraint of A can be defined because A has a finite repetitive precedence constraint between its three successive repetitions, and this precedence constraint is completed by the infinitive repetitive precedence constraint. The periodicity constraint of B can be defined only thanks to the infinite repetition of the pattern. The operation C can not have a periodicity constraint because of the potential parallelism of C_1 and C_2 which does not impose an order of execution between C_1 and C_2 each time the pattern is repeated. Then we can not have the notion of consecutive repetitions necessary to define the periodicity of C.

The problem to be solved is the following: for the considered system modeled by a graph with repetitive precedence constraints (repetitive graph), and with latency and periodicity constraints, we must find a feasible schedule. That is to say, a schedule which gives the start time for all the operations and which satisfies the real-time constraints for the problem: [*1/ repetitive graph/ periodicity and latency constraints*].

4 Algorithm

This section is organized as follows: we present the algorithm for the problem [*1/ repetitive graph/ periodicity and latency constraints*] and we show that it is optimal (if there is a feasible schedule for the problem, the algorithm will find it). In order to solve the problem, we first give an algorithm of latency marking for the vertices of the pattern, and then using these latency marks we give the scheduling algorithm which is a non-preemptive, off-line algorithm. This latter algorithm is applied to the infinitely repeated pattern.

4.1 Algorithm of latency marking

We denote by $G = (V, E)$ the directed acyclic graph of operations with repetitive precedence constraints, where V is the set of vertices and $E \subseteq V \times V$ the set of edges, and by $\mathcal{D}(A) = \{B \in V$ such that $\exists$ a directed path starting with A and ending with $B\}$.

Given that some pairs of operations have a latency constraint, the goal of the algorithm is to assign to each operation A a number. This number indicates if an operation C, belonging to a pair (B, C) with a latency constraint, will be executed after the execution of A due to the existence of a directed path between A and C, and no directed path between A and B. If for an operation, there are several operations satisfying this property, then the number will be the smallest value of these latency constraints. We call this number $mark(A) \in \mathbb{N}^* \bigcup \{\infty\}$ (∞ is a natural number bigger than all the natural numbers). The latency marks evolve during the algorithm of latency marking, which is applied to the pattern.

Lemma 4.1 *If a pair of operations* (A, B) *has a latency constraint* $L(A, B)$ *and if there is an operation* C *with* $A \in \mathcal{D}(C)$*, then* $s_B + C_B - s_A \leq L(A, B)$ *is satisfied* $\forall s_C$.

Proof If (A, B) has a latency constraint, then $B \in \mathcal{D}(A)$ and $s_A \leq s_B$. Because $A \in \mathcal{D}(C)$ we have $s_C \leq s_A$. Then we have $s_C \leq s_A \leq s_B$ which means that because C is already scheduled when A becomes schedulable, the start time of the operation B is not affected by the start time of C. □

Algorithm

Initialization: If (A, B) has a latency constraint $L(A, B)$ then $mark(B) = L(A, B)$ and $mark(A) = \infty$, otherwise $mark(A) = mark(B) = \infty$. Moreover, if B belongs to several pairs of operations having different latency constraints, then $mark(B) = min_{(C,B) \in \mathcal{L}}\{L(C, B)\}$, and $mark(A) = \infty$. We denote by $\mathcal{W}$ the working-set and let $\mathcal{W} = \mathcal{L}$.

Step 1: for $(A, B) \in \mathcal{W}$ and for each operation $C \in V \backslash \{A, B\}$, we have three possibilities:
(a) if $A \in \mathcal{D}(C)$, then according to Lemma 4.1, $mark(C) = mark(C)$;
(b) if $A \notin \mathcal{D}(C)$ and $B \in \mathcal{D}(C)$, then $mark(C) = min(mark(C), mark(B))$;
(c) if $A \notin \mathcal{D}(C)$ and $B \notin \mathcal{D}(C)$, then $mark(C) = mark(C)$.
The pair (A, B) is removed from $\mathcal{W}$.

Step 2: if $\mathcal{W} \neq \emptyset$ then goto step 1, otherwise the algorithm stops.

We apply the algorithm of latency marking to the pattern of the example depicted figure 4. At the beginning of the algorithm, we have the working-set $\mathcal{W} = \{(B, C_1), (A_2, C_2)\}$. The table 1 gives the values of the functions $mark$ for each operation of the pattern. The final values of the functions $mark$ are given by the last line of this table. These are the values that will be used by the scheduling algorithm.

4.2 Scheduling algorithm

The scheduling algorithm transforms the partial order associated to the graph in a (one of the possible) total order satisfying the constraints. This algorithm is applied to the in-

	$mark(A_1)$	$mark(A_2)$	$mark(A_3)$	$mark(B)$	$mark(C_1)$	$mark(C_2)$
Initialization	∞	∞	∞	∞	9	10
$L(B, C_1)$	∞	∞	∞	∞	9	10
$L(A_2, C_2)$	∞	∞	10	10	9	10

Table 1: Results given by the algorithm of latency marking

finitely repeated pattern.

We denote by $\mathcal{W}$ the working-set, by s_l the start time of the last operation that was scheduled, by C_l its computation time, and by $\mathcal{P}$ the set of all the operations which have a periodicity constraint. During the algorithm $\mathcal{W}$ contains all the *schedulable operations*, i.e. operations the predecessors of which are already scheduled. We note that between two operations which schedulable in the same time there is no path. Implicitly, every time an operation is scheduled s_l (resp. C_l) changes its value into the start time (resp. computation time) of this operation.

Algorithm

Initialization: $\mathcal{W} = \bigcup_{A \in V \ and \ Prec(A)=\emptyset} \{A\}$ and $s_l = 0, C_l = 0$.

Step 1 *(operation with latency constraints)*:
if $\exists A \in \mathcal{W}$ such that $mark(A) \neq \infty$ then $s_A = s_l + C_l$ such that $mark(A) = min_{B \in \mathcal{W}}\{mark(B)\}$, we remove it from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 5.

Step 2 *(operation without periodicity and which are not first in a latency constraint)*:
if $\exists A \in \mathcal{W}$ such that there is no $B \in V$ with $(A, B) \in \mathcal{L}$ and $A \notin \mathcal{P}$, then $s_A = s_l + C_l$, we remove it from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 1.

Step 3 *(operation which are first in a latency constraint)*:
if $\exists A \in \mathcal{W}$ such that there is $B \in V$ with $(A, B) \in \mathcal{L}$, then $s_A = s_l + C_l$ with $L(A, B) = max_{(C,D) \in \mathcal{L} \ and \ C \in \mathcal{W}}\{L(C, D)\}$, we remove it from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 5.

Step 4 *(operation with periodicity for which each first repetition is not already scheduled)*:
we have $s_A = s_l + C_l$ such that $T_A = min_{C \in \mathcal{W} \bigcap \mathcal{P} \ and \ C_0 \ not \ already \ scheduled}\{T_C\}$. Go to Step 6.

Step 5: if $\nexists A \in \mathcal{W} \bigcap \mathcal{P}$ with A_1 already scheduled, then go to Step 1.

Step 6: we search an operation $A \in \mathcal{W} \bigcap \mathcal{P}$ for which its first repetition A_1 is scheduled and for which we have:

$$s_{A_i} + T_A - s_l - C_l = min_{B_i \in \mathcal{W} \bigcap \mathcal{P} \textit{ and } B_0 \textit{ already scheduled}}\{s_{B_i} + T_B - s_l - C_l\}$$

where A_i is the last repetition scheduled of A. If we find several operations A then the system is not schedulable and the algorithm stops.

Step 7 *(operation with latency constraints)*:
if $\exists C \in \mathcal{W} \backslash \{\{A\} \bigcap P\}$ such that $s_l + C_l + C_C \leq s_{A_i} + T_A$ for the operation A found previously, then we have $s_C = s_l + C_l$ with: $mark(C) = min_{D \in \mathcal{W} \textit{ and } C_D \leq T_A}\{mark(D)\}$, the scheduled operation is removed from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 6.

Step 8 *(operation with periodicity for which each first repetition is not already scheduled)*:
if $\exists C \in \mathcal{W} \bigcap \mathcal{P}$ with C_0 not scheduled and $s_l + C_l + C_C \leq s_{A_i} + T_A$ for the operation A found previously, then we have $s_C = s_l + C_l$ with $T_C = min_{D \in \mathcal{W} \bigcap \mathcal{P} \textit{ and } C_D \leq T_A}\{T_D\}$, the scheduled operation is removed from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 6.

Step 9 *(operation with periodicity)*:
we have $s_A = s_l + C_l$ such that $s_{A_{i+1}} = s_{A_i} + T_A$, the scheduled operation is removed from $\mathcal{W}$, all the operations which became schedulable are added to $\mathcal{W}$, and go to Step 6.

Remark 4.1 *the scheduling algorithm never stops unless the periodicity constraints can not be satisfied in the Step 6, i.e. in this case, the system is not schedulable.*

We apply the scheduling algorithm to the example of the figure 4 for which we have $\mathcal{P} = \{A, B\}$. The beginning of the results are given in the table 2.

Lemma 4.2 *The periodicity constraint is a particular case of a latency constraint. Consequently, the periodicity constraint is a stronger constraint than the latency constraint.*

Proof If A has a periodicity constraint we have $s_{A_{i+1}} - s_{A_i} = T_A, \forall i \in \mathbb{N}$. If the pair $(A_i, A_{i+1}), \forall i \in \mathbb{N}$ has a latency constraint L, then $s_{A_{i+1}} - s_{A_i} + C_{A_{i+1}} \leq L$. After replacing $L - C_{A_{i+1}}$ by L', we have $s_{A_{i+1}} - s_{A_i} \leq L'$. Then we notice that the equality expressing the periodicity constraint is included in the inequality expressing the latency constraint. $\Box$

Remark 4.2 *The latency constraint of the pair (A_i, A_{i+1}) implies that A_{i+1} may be executed after the execution of A_i, anytime during the interval $(s_{A_i}+C_{A_i}, s_{A_i}+C_{A_i}+L)$. But, if A has a periodicity constraint the start time of A_{i+1} must be equal to $s_{A_i} + T_A, \forall i \in \mathbb{N}$.*

Remark 4.3 *We denote four types of operations to be scheduled, during the algorithm, as follows:*
(k_1) operations A with $mark(A) \neq \infty$;
(k_2) operations A with $mark(A) = \infty$ and there is not $B \in V$ with $(A, B) \in \mathcal{L}$ and

$\mathcal{W}$	s_l	C_l	Step used	Action
$\{A_1\}$	0	0	Step 4	$s_{A_1} = 0$
$\{A_2\}$	0	2	Step 6	A chosen
$\{A_2\}$	0	2	Step 9	$sA_2 = 5$
$\{A_3\}$	5	2	Step 6	A chosen
$\{A_3\}$	5	2	Step 9	$s_{A_3} = 10$
$\{A_1, B\}$	10	2	Step 6	A chosen
$\{A_1, B\}$	10	2	Step 8	$s_B = 12$
$\{A_1, C_1, C_2\}$	12	1	Step 6	A chosen
$\{A_1, C_1, C_2\}$	12	1	Step 7	$s_{C_1} = 13$
$\{A_1, C_2\}$	13	2	Step 6	A chosen
$\{A_1, C_2\}$	13	2	Step 9	$s_{A_1} = 15$
$\{A_2, C_2\}$	15	2	Step 6	A chosen
$\{A_2, C_2\}$	15	2	Step 7	$s_{C_2} = 17$
$\{A_2\}$	17	2	Step 6	A chosen
$\{A_2\}$	17	2	Step 9	$s_{A_2} = 20$
$\{A_3\}$	20	2	Step 6	A chosen
$\{A_3\}$	20	2	Step 9	$s_{A_3} = 25$
$\{A_1, B\}$	25	2	Step 6	B chosen
$\{A_1, B\}$	25	2	Step 9	$s_B = 27$
$\{A_1, C_1, C_2\}$	27	1	Step 6	A chosen
...				

Table 2: Results given by the scheduling algorithm

$A \notin \mathcal{P}$;
(k_3) operations A with $mark(A) = \infty$ and $\exists B \in V$ with $(A, B) \in \mathcal{L}$;
(k_4) operations A with $mark(A) = \infty$ and $A \in \mathcal{P}$.
Note that the operations of type k_2 do not have any constraint, and they are scheduled only if there is no more operation with constraints among the schedulable operations.

Theorem 4.1 *For a system of operations with precedence and periodicity constraints and without latency constraint, a schedule obtained according to the increasing order of the periodicity constraints of the first instances of the schedulable operations, is feasible if and only if it is feasible according to the decreasing order of the periodicity constraints of the first instances of the schedulable operations.*

Proof We demonstrate the equivalence by double implication. First, we demonstrate that if the system is schedulable according to the increasing order of the periodicity constraints of the first instances of the schedulable operations, then the system is, also, schedulable according to the decreasing order of the periodicity constraints of the first instances of the schedulable operations. We can demonstrate the implication for the two first instances of the operations with the smallest values of the periodicity constraints and then by mathematical induction the result can be generalized to all the first instances of the schedulable operations. We denote by A and B the two operations with the smallest values of the periodicity constraints and we suppose $T_A \leq T_B$. We have $s_{A_i} = s_{A_1} + iT_A, \forall i > 1$, and similarly, $s_{B_j} = s_{B_1} + jT_B, \forall j > 1$. Because the system is schedulable, we have $s_{A_i} \neq s_{B_j} \forall j < i$ and $C_A + C_B \leq T_A$. Moreover, because it is schedulable according to the increasing order of the periodicity constraints of the first instances of the schedulable operations, we have:

$$s_{B_1} = s_{A_1} + C_A \Rightarrow \begin{cases} s_{A_i} = s_{A_1} + iT_A \\ s_{B_j} = s_{A_1} + C_A + jT_B \end{cases}$$

From the previous relations, we have $C_A + jT_B \neq jT_A$ and $C_A + jT_B > jT_A, \forall j \in \mathbb{N}$. We demonstrate by contradiction that we obtain also a feasible schedule if we schedule the operations according to the decreasing order of the periodicity constraints of the first instances of the schedulable operations. Indeed, in this case, we have:

$$s_{A_1} = s_{B_1} + C_B \Rightarrow \begin{cases} s_{A_i} = s_{B_1} + C_B + iT_A \\ s_{B_j} = s_{B_1} + jT_B \end{cases}$$

Therefore, we assume that we do not obtain a feasible schedule, i.e. $s_{A_i} = s_{B_j}, \forall j > i$. We have also:

$$\left. \begin{array}{l} C_B + iT_A = jT_B \\ jT_B > jT_A - C_A \end{array} \right\} \Rightarrow C_A + C_B > (j-i)T_A, \forall j > i$$

which is in contradiction with $C_A + C_B \leq T_A$. The second part of the demonstration is similar to the first one.$\square$

Theorem 4.2 *The scheduling algorithm 4.2 is optimal (if there is a feasible schedule, the algorithm will find it).*

Proof We consider three cases:

1. the system has only precedence and latency constraints. Then we schedule a system that may have operations of types k_1, k_2 and k_3 (Remark 4.3). At the beginning of the algorithm we have two possibilities:

   (a) there are operations of type k_1 among the schedulable operations. It means that along the algorithm only the Step 1 schedules the operations, and this, according to the increasing order of their marks. We prove the optimality of this Step 1 by contradiction. For this, we assume that the operations are scheduled according to the decreasing order of their marks, i.e. for two operations A and B if $mark(A) < mark(B)$, then $s_B < s_A$. We consider the latency constraint $L(E, C)$ with $C \in \mathcal{D}(A)$ and $L(E, C) = mark(A)$ (c.f. algorithm of latency marking, Step 1, b)). Also we consider the latency constraint $L(F, D)$ with $D \in \mathcal{D}(B)$ and $L(F, D) = mark(B)$. Finally, we obtain $s_D < s_C$ which is in contradiction with the fact that $L(E, C) < L(F, D)$.

(b) there is no operation of type k_1 among the schedulable operations. It means that the schedule is obtained by using the Step 2 or the Step 3, until an operation of type k_1 becomes schedulable. The Steps 2 and 3 schedule an operation A of type k_2 before an operation B of type k_3. When the operation B is of type k_3, then $\exists C \in V$ such that the pair $(B, C) \in \mathcal{L}$ and because $mark(A) = \infty$, we have $C \notin \mathcal{D}(A)$. It means that there are two possibilities for the schedule, either $s_A < s_B < s_C$, either $s_B < s_A < s_C$. We observe that in both cases, the start time of operation A does not modify the value of $s_C - s_B$, and consequently the latency constraint. So, it does not matter what is the scheduling of the operations of type k_2 and k_3. Once an operation of type k_3 is scheduled, operations of type k_1 become schedulable and, until the algorithm stops only the Step 1 schedules the operations. Hence we are in the case (a), and we saw previously that the choice made by Step 1 is optimal.

2. the system has precedence and periodicity constraints and no latency constraints. Then we schedule a system that may have only operations of type k_4 (Remark 4.3). In this case because $mark(A) = \infty, \forall A \in \mathcal{W}$, the schedule is found only by using the steps 4, 8 and 9. These steps schedule the schedulable operations in the increasing order of their periodicity constraints. Indeed, due to the theorem 4.1 it is equivalent to schedule the first instances of the operations, with periodicity constraints, according to the increasing or to the decreasing order of their periodicity constraints. Once the first instances of all the operations with periodicity constraints were scheduled, the scheduling algorithm only have to schedule the operations according to their periodicity constraints.

3. the system has the three constraints: precedence, periodicity and latency. At the beginning of the algorithm we have two possibilities:

   (a) there are operations of type k_1 among the schedulable operations. An operation with $mark \neq \infty$ may have or not a periodicity constraint. The periodicity constraint must be satisfied, only after the first repetition of the operation is scheduled. As soon as Step 1 scheduled the first repetition of an operation with periodicity constraint, the remaining operations are scheduled only by using the Steps 8, 9 and 10. So, until the first repetition of an operation with a periodicity constraint is scheduled, only the Step 1 is used. We have two possibilities:

      i. no first repetition of an operation with a periodicity constraint was yet scheduled. It means that we have to satisfy only latency constraint and we are in same situation as 1.(a), and we saw that the choice made by Step 1 is optimal.

      ii. the first repetition of an operation with a periodicity constraint was scheduled, so we have a periodicity constraint to satisfy. After calculating the start time of the operation A such that

$$s_{A_i} + T_A - s_l - C_l = min_{B_i \in \mathcal{W} \bigcap \mathcal{P} \textit{ and } B_0 \textit{ already scheduled}} \{s_{B_i} + T_B - s_l - C_l\}$$

(Step 7), the remaining operations are scheduled by using the Step 8, 9 and 10. The Steps 8 and 9 search for operations, among the schedulable operations, that may be scheduled before the calculated start time. Because we already chose the first periodicity constraint to be satisfied, we have only latency constraints to meet like in the case 1.(a), and we saw that this choice of the operations with the smallest $mark$ is optimal. If there is no operation with $mark \neq \infty$ which may be scheduled before the calculated start time, then it does not matter how the remaining operations are scheduled before the calculated start time. Finally, the Step 10 schedules the operation A at the calculated start time.

(b) there is no operation of type k_1 among the schedulable operations. It means that the schedule is obtained by using the Steps 2, 3 and 4 until an operation with periodicity and/or type k_1 becomes schedulable. The Step 3 is used only if there is no more operation of type k_2. As soon as the Step 3 scheduled an operation, operations of type k_1 become schedulable and the Step 4 will never be used. If the Step 4 schedules an operation, then only the Steps 8, 9 and 10 will schedule the remaining operations. So, the Steps 2, 3 and 4 schedule either the operations of type k_2 before scheduling operations of type k_3, either the operations of type k_2 before scheduling operations of type k_4. We saw in 1.(b) that it does not matter how the operations of type k_2 and k_3 are scheduled. An operation of type k_2 has no constraint to satisfy, and an operation of type k_4 once its first repetition is scheduled, has its periodicity constraint to satisfy. Hence, for operations of type k_2 and k_4, we reduce the number of operations which must be scheduled when there are periodicity constraints to be satisfied. Once the Step 3 or the Step 4 is used, we are in the case i or ii. $\square$

4.3 Consequence of solving our problem

We remind that we denote by D_A the deadline of an operation A defined from the beginning of the schedule implying that $s_A \leq D_A$.

Theorem 4.3 *The latency constraint $L(A, B)$ of a pair $(A, B) \in \mathcal{L}$ is equivalent to the deadline of operation B.*

Proof We demonstrate the equivalence by double implication.
First, we demonstrate that a latency constraint $L(A, B)$ of a pair $(A, B) \in \mathcal{L}$ may be expressed as the deadline of operation B, once A is scheduled. Because $(A, B) \in \mathcal{L}$, then we have $s_B \leq L(A, B) + s_A - C_B$. If we denote by D_B the expression $L(A, B) + s_A - C_B$, we obtain $s_B \leq D_B$. Therefore, as soon as A is scheduled, the value of D_B is known.

Second, we express the deadline of an operation B as a latency constraint $L(\cdot, B)$. Because B has a deadline D_B, then $s_B \leq D_B$. We denote by A the first operation which was scheduled, then $s_A = 0$. Because the deadline is defined from the start of the schedule, we obtain that $s_B - s_A \leq D_B$. Moreover, because all the operations without predecessors may be the first scheduled operation, we have $s_B - s_C \leq D_B, \forall C \in V$ with $Prec(C) = \emptyset$

and $B \in \mathcal{D}(C)$. So, in order to satisfy the deadline of B, we define several latency constraints $L(C, B) = D_B$ for each C such that $Prec(C) = \emptyset$ and $B \in \mathcal{D}(C)$. $\square$

Due to this theorem, we may use the same scheduling algorithm in order to solve the problem [*1/ repetitive graph/ periodicity constraints and deadlines*], but in this case the function $mark$ of the operation A must be equal to the $min_{B\in\mathcal{L}(A)}\{D_B\}$. The following algorithm of deadline marking allows to obtain these marks:

Algorithm

Initialization: $\mathcal{W} = \bigcup_{A\in V \ and \ Suc(A)=\emptyset} Prec(A)$ is the working-set. If A has a deadline D_A, then $mark(A) = D_A$, otherwise $mark(A) = \infty$.

Step 1: For $A \in \mathcal{W}, mark(A) = min(mark(A), min_{B\in Suc(A)}\{mark(B)\})$ and $\mathcal{W} = \mathcal{W}\backslash\{A\}$. We add to $\mathcal{W}$ the operations for which all the predecessors has been removed from $\mathcal{W}$.

Step 2: Repeat Step 1 until $\mathcal{W} = \emptyset$.

Remark 4.4 *recursively, each operation inherits the deadline of its successors. At the end of this latter algorithm, the mark of an operation may be equal to either its inherited deadline or its initial deadline. Therefore, the algorithm does not modify the marks of the operations without successors. This algorithm is, also, applied to the pattern.*

5 Conclusion and further research

The paper gives a model based on graphs in order to state clearly the problem of scheduling real-time systems with precedence, periodicity and latency constraints for one computing resource. Because usually the problem of scheduling systems with periodicity constraints and deadlines, and the problem of scheduling systems with precedence constraints and deadline, are treated separately, we define the notion of latency constraint. By proving its equivalence with the notion of deadline which is commonly used, we merge these two domains of research. We give an optimal algorithm for solving our problem, and in order to prove the optimality of this algorithm we need to demonstrate that in the case of systems without latency constraint, it is equivalent to schedule the first instances of the operations with periodicity constraints, according to the increasing or to the decreasing order of their periodicity constraints. These results have as consequence the extension of the algorithm for scheduling real-time systems with precedence, periodicity constraints and deadlines for one computing resource.

Presently we are searching for a condition establishing if a real-time system with precedence, periodicity and latency constraints, is schedulable. Moreover, when this condition is not satisfied, we plan to study if introducing preemption allows to find a feasible schedule. Finally, we plan to extend the problem for several computing resources and to calculate its complexity.

References

[1] E.L. Lawler. Recent results in the theory of machine scheduling. *Mathematical Programming: the State of the Art, Springer-Verlag*, 1983.

[2] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[3] J.P. Lehoczky, L. Sha, and Y Ding. The rate monotonic sheduling algorithm: exact characterization and average case bahavior. *Proceedings of the IEEE Real-Time Systems Symposium*, 1989.

[4] N.C. Audsley, A. Burns, M.F. Richardson, Tindell K., and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineerung Journal*, 1993.

[5] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real time embedded heterogeneous multiprocessors. *Codes'99 7th International Workshop on Hardware/Software Co-Design*, 1999.

[6] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. Technical report, University of California, 1971.

[7] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 1973.

[8] J.K. Lenstra and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.*, 1977.

[9] T.P. Baker. Stack-based scheduling of realtime processes. *The Journal of Real-time Systems*, 1991.

[10] J. Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines. *Modelling and Performance Evaluation of Computer Systems*, 1976.

[11] David Harel and Amir Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*. Springer Verlag, New York, 1985.

[12] S.K. Baruah, R.R. Howell, and L.E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Journal of Real-Time Systems*, 1990.