The impact of core precedences in a cyclic RCPSP with precedence delays

Zdenek Hanzalek, Claire Hanen

Journal of Scheduling manuscript No.
(will be inserted by the editor)

Impact of Core Precedences in a Cyclic RCPSP with Precedence Delays

Z. Hanzalek · C. Hanen

the date of receipt and acceptance should be inserted later

Abstract In this paper, we introduce new kind of constraints, called core precedence constraints, in a cyclic Resource Constraint Project Scheduling Problem with precedence delays. We show, by an example, which kind of industrial constraints might be modeled by such core precedences in a periodic production setting. We then establish that these constraints can be quite easily added to ILP formulation of the cyclic RCPSP. Although core precedences seem to be very similar to classical precedence, they can induce infeasibility even without resource constraints. Moreover, we show that the feasibility checking problem is NP-complete in the strong sense, even assuming unit processing times and no resource constraints.

1 Introduction

A cyclic extension of the Resource Constraint Project Scheduling Problem with precedence delays (RCPSP) is a useful way to model scheduling problems occurring in code generation for VLIW architectures (Very Long Instruction Word refers to a processor architecture designed to take advantage of instruction level parallelism)[27][9] as well as in production systems [25][22]. RCPSP [18] was also used to model a message scheduling problem [15] in industrial communication protocols [17].

Cyclic scheduling problems of iterative loops or production processes are induced by a set of computing or manufacturing operations to be repeated. An iteration is defined by one repetition of this set. A periodic schedule starts a new iteration every period (so a period reflects the takt time). Iterations are usually interleaved to achieve efficiency : once started an iteration may span over several periods, so that in each period interval, iteration numbers of tasks are not necessarily the same.

Zdenek Hanzalek
Czech Technical University in Prague, Faculty of Electrical Engineering
E-mail: hanzalek@fel.cvut.cz

Claire Hanen
LIP6, UPMC, Paris, and University of Paris-Ouest-Nanterre-La-Défense
E-mail: claire.hanen@lip6.fr

In [16] the authors studied the scheduling problems induced by the IEEE 802.15.4/ZigBee communication protocol and they observed that this problem has a cyclic scheduling nature. This paper was first motivated by the use of algorithms developed for cyclic RCPSP to solve this problem. However, our study led us to introduce, within the cyclic RCPSP framework, the new concept of core precedences.

In a cyclic RCPSP setting the concept of uniform precedences [27] is used to control interleaving of iterations. However, uniform precedence constraints and RCPSP resource constraints (see the definition and an example in Section 2.1) are not sufficient to describe the whole ZigBee problem [16], since they are not able to model some relations between tasks within the period, regardless of their iteration number. In this paper we show other practical examples (e.g. cleaning of a machine before the end of a shift) where such particular precedence constraints, called core precedences, are needed.

However, core precedences may induce infeasibility of the whole system. Hence, in this paper, we study the feasibility checking problem with and without resource constraints. We first show how core precedence constraints can be included in an ILP model for the feasibility problem with resource constraints and with a fixed period.

Furthermore, we show that the decomposed software pipelining approach described in [5] can be used to reformulate the feasibility checking problem as a graph problem. This formulation is used to prove that combining uniform precedences and core precedences leads to an $\mathcal{NP}$-complete problem in the strong sense, even if unit processing times are assumed and resource constraints are omitted.

This paper is organized in 5 sections. Section 2 describes the problem settings and introduces core precedence constraints through an example and core feasibility problems. In Section 3, a short discussion on the extension of previous approaches led us to introduce some useful notations and definitions from which a graph model for the problem without resource constraints is set. Then, we show an ILP model of the problem with a fixed period and resource constraints. Section 4 is devoted to the complexity of the feasibility problem without resource constraints. We prove its $\mathcal{NP}$-completeness by a reduction from 3-SAT. The last section concludes the paper.

2 Problem Definition

In this section, we first introduce the cyclic RCPSP setting [9]. Then, we introduce core precedences and we show how they can be used to model practical problems. We then state and discuss the feasibility problems that are induced by these new constraints.

2.1 Cyclic RCPSP based on uniform precedences

Among models for cyclic scheduling problems with temporal and resource constraints [14], [27], [22], [1], the Cyclic unitary RCPSP problem is one of the most useful, for which some approximation and ILP algorithms have been provided and evaluated on benchmarks [5], [9], [2]. In this subsection, we recall the main features of this model, then in next subsection we show how it can be extended to handle the special constraints of our motivating example.

2.1.1 Statements and notations

In this model, the set of tasks $\mathcal{T}$ is supposed to be executed an infinite number of times (consider for example infinite execution of an iterative loop). For each task T_i, and each iteration number q, we denote by T_i^q the q^{th} occurrence of task T_i.

An infinite schedule s assigns a starting time s_i^q to each occurrence T_i^q of task T_i.

In the rest of the paper, we focus on periodic schedules, i.e. schedules such that each task is repeated every λ time unit: $\forall\, T_i \in \mathcal{T}, \quad s_i^q = s_i + (q-1)\lambda$, where $s_i = s_i^1$ is the starting time of the first occurrence of T_i, and λ is (the length of) the period of the schedule.

Remark 1 The term "periodic scheduling" is often related to the "multi-periodic" problems where the required periods of tasks are different [20]. But in cyclic scheduling problems [24], we use the term "periodic " when each task is repeated every λ time units, though all tasks have the same period.

The resource constraints are defined as follows:

1. m resource types. The availability of resource type $k \in \{1, \ldots, m\}$ is denoted by R_k.
2. A set $\mathcal{T}$ of n tasks $\{T_i,\}_{1\leq i\leq n}$ with integer processing times $\{p_i,\}_{1\leq i\leq n}$. Each occurrence of T_i uses r_{ik} units of type k resource during its execution, where r_{ik} is an integer.
3. If s is a schedule, then for any time unit t, and any resource k, the sum of resource requirements made by all tasks T_i^q, such that $s_i^q \leq t < s_i^q + p_i$, is not greater than R_k.
4. The problem is called unitary if $r_{ik} \in \{0, 1\}$.

A widely used model for cyclic precedence constraints is called "uniform precedences" [14]. Such constraints are defined by a bi-valued graph $G_U = (\mathcal{T}, \mathcal{E}_{\mathcal{U}})$ with nonnegative integer valuations l and h called, respectively, the length and height of the arc. For arc $a = (T_i, T_j)$ we denote its length by l_{ij} and its height by h_{ij}. The length denotes a precedence delay while the height counts the number of iterations by which i precedes j (see below). We assume here that l and h are nonnegative.

An arc $a = (T_i, T_j)$ of G_U induces an infinite number of precedence constraints on any schedule s as follows: for any iteration q, T_i^q precedes $T_j^{q+h_{ij}}$ by the precedence delay l_{ij} i.e.

$$s_j^{q+h_{ij}} - s_i^q \geq l_{ij}$$

Notice that for a periodic schedule, a uniform precedence induces a simple linear inequality [14] :

$$s_j - s_i \geq l_{ij} - \lambda h_{ij}$$

For path μ of G_U, we denote the sum of the lengths of its arcs by $L(\mu)$, and the sum of the heights of its arcs by $H(\mu)$. The example below shows a classical application for which uniform precedences define a convenient model.

2.1.2 Example of uniform precedences

Let us consider a signal processing application with a periodically sampled input U and output Y realized by the loop shown below.

```
for q := 1 to N do
    a(q) := U(q) - c(q-2); // task T1
    b(q) := a(q) * 5; // task T2
    c(q) := b(q) + U(q); // task T3
    d(q) := b(q) + c(q-2); // task T4
    Y(q) := U(q-1) + d(q); // task T5
end
```

The application is executed on two addition units and one pipelined multiplication unit. All tasks have a unit processing time. The result on the (non-pipelined) addition unit is available immediately after processing, therefore the corresponding $l_{ij} = 1$. The result on the pipelined multiplication unit is available with one time unit delay, therefore, the corresponding $l_{ij} = 2$. The height h_{ij} specifies the shift of the iteration related to the data produced by task T_i and read (consumed) by task T_j. The corresponding uniform graph is shown on Figure 1.

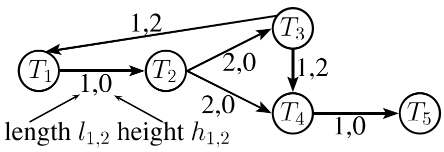

Fig. 1 Uniform graph

The resulting cyclic schedule of the first, second and third iteration for the loop with $N = 3$ is shown on Figure 2. This example illustrates a typical property of a periodic cyclic schedule: an iteration spans over several periods (e.g. the first iteration is scheduled in period 1, 2 and 3). Consequently, a period contains tasks from different iterations (e.g. period 3 contains tasks from the first, second and third iteration).

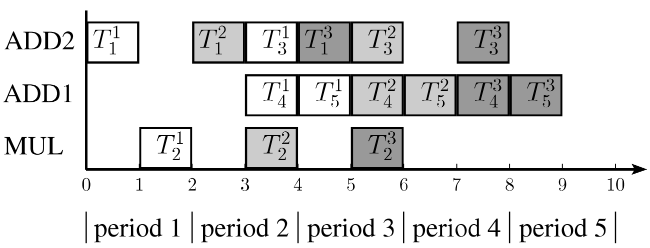

Fig. 2 Periodic schedule

2.2 Core of a periodic schedule and core precedence

Let s be a periodic schedule of period λ. It is useful to consider another representation of s, that will be used in the next section.

For a task T_i, we decompose s_i, the starting time of its first occurrence, modulo the period λ: $s_i = \sigma_i + \alpha_i\lambda$, with $\sigma_i \in [0, \lambda), \alpha_i \in \mathbb{N}$. We call α_i the retiming of T_i, and the vector σ is called the core of the periodic schedule. Indeed, we can observe that in any feasible periodic schedule with period λ, if we consider an interval $[q\lambda, (q+1)\lambda)$ of the steady state (for a large enough q), then $q\lambda + \sigma_i$ is the starting time of the only occurrence of T_i in this interval. This occurence is then $T_i^{q+1-\alpha_i}$.

Thus, σ_i is an acyclic schedule of tasks that fulfills the resource constraints. The retiming α_i models the span of an iteration over different periods in the schedule.

For example, in the case of the cyclic schedule shown on Figure 2: $\lambda = 2$, $\sigma_2 = 1$, $\alpha_2 = 0$, as T_2^1 is scheduled in period 1, and $\sigma_5 = 0$, $\alpha_5 = 2$, as T_5^1 is scheduled in period 3.

Notice that a uniform precedence constraint (T_i, T_j) induces the following inequality:

$$\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i) \tag{1}$$

We now introduce a new type of constraint, called core precedences, aimed at modeling some constraints observed in practice, that bind tasks within the core schedule, regardless of the retiming of the underlying schedule:

Consider an instance of the cyclic RCPSP problem described in previous section. A core precedence constraint links the core starting times of two tasks T_i and T_j with a time lag denoted by $c_{ij} \in \mathbb{Z}$

$$\sigma_j - \sigma_i \geq c_{ij} \tag{2}$$

We consider a graph $G_C = (\mathcal{T}, E_C)$ of such core constraints in addition with an instance of the cyclic RCPSP problem.

Notice that a core precedence links the cores of tasks occurrences within any period interval of the steady state of a periodic schedule, regardless of their iteration number. So, it shares this feature with usual resource constraints. However there are also some differences. For example let us consider a disjunctive constraint between two tasks T_i and T_j. Then, in any period interval of a periodic schedule the current occurrence of these two tasks should not be computed simultaneously, regardless of the iteration number of these two occurrences. So the scheduler has to decide whether the occurrence of T_i is scheduled before or after the occurence of T_j in the core schedule. A core precedence constraint does not give such a choice: the order between the two tasks is induced by the constraint. However, a core constraint could result from partial choices made on usual disjunctions.

2.3 Example of core precedences

As a motivation example, we introduce a lacquer production scheduling problem inspired by an industrial case study [4], introduced in the Ametist project. We assume a cyclic production of bronze (tasks T_1 to T_3 in Figure 3), metallic (tasks T_4 to T_8), and universal (tasks T_9 to T_{12}) lacquers with a fixed volume to be produced each week. We deal with a fixed period of one week, where working days are used for production and weekend is used for a general clean-up. The recipe for a given type and quantity of lacquer defines the production tasks with the processing times, required resources (see Table 1), and uniform precedence constraints (solid arcs labeled l_{ij}, h_{ij} in Figure 3).

Five production steps are assumed: In the first step, solid and liquid input materials and solvents are prepared (tasks T_1, T_5, T_6, T_9 and T_{10}). Then, in the second step, the materials are unified using dose spinner (tasks T_2, T_7 and T_{11}). When the unification procedure is finished, the lacquer quality is checked in the laboratory. The laboratory check takes 60 time units for bronze lacquer and 100 time units for metallic or universal lacquer. These operations are modeled by arc lengths since the laboratory is an unconstrained resource. The arc length relates the starting time of the origin task to the starting time of the destination task. Therefore, it includes processing time of the origin task and duration of the laboratory check. For example, $l_{23} = 8 + 60, l_{78} = 12 + 100$ and $l_{11,12} = 10 + 100$. Finally, the mixing vessel is emptied at the filling station (tasks T_3, T_8 and T_{12}). When the filling operation is finished, the mixing vessel is cleaned. The cleaning of the vessel is performed by an unconstrained resource in 4 time units (modeled by the arc lengths including the processing time of the origin task and duration of the cleaning $l_{31} = 12 + 4, l_{84} = 12 + 4$ and $l_{12,9} = 14 + 4$).

There are the following resources:

- resource 1 : mixer with capacity $R_1 = 1$,
- resource 2 : dispersing line with capacity $R_2 = 2$,
- resource 3 : dose spinner with capacity $R_3 = 1$,
- resource 4 : filling station with capacity $R_4 = 1$,
- and resource 5 is a fictive resource with capacity $R_5 = \infty$ to execute dummy tasks.

Different vessels are used for the production of lacquers: two for the bronze one, three for the metallic one, and two for the universal lacquer. The available vessels are modeled as nonzero heights of corresponding uniform precedence constraints (e.g. $h_{31} = 2$ in the case of the bronze lacquer). Dummy task T_4 is used to model the start of the metallic lacquer production.

task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	$T_{14\ to\ 17}$	T_{18}
proc.time	8	8	12	0	8	14	12	12	8	16	10	14	0	8	56
resource	1	3	4	5	1	2	3	4	1	2	3	4	5	5	5

Table 1 The processing times and required resources for tasks in Figure 3

Core constraints are represented by dashed arcs in Figure 3 (the length of an arc equals zero when there is no label). They are used to model the following requirements independent of the iteration number:

- In order to absorb blend (a mixture of two types of lacquer), the tasks performed on the filling station should be executed such that the previous lacquer is absorbed by the subsequent one, i.e. in our case, the bronze one is filled first, then the metallic one and finally the universal one. A core precedence constraint relates the core starting time of the origin task to the core starting time of the destination task. Therefore, the time lag is equal to the processing time of the origin task ($c_{38} = 12$ and $c_{8,12} = 12$).

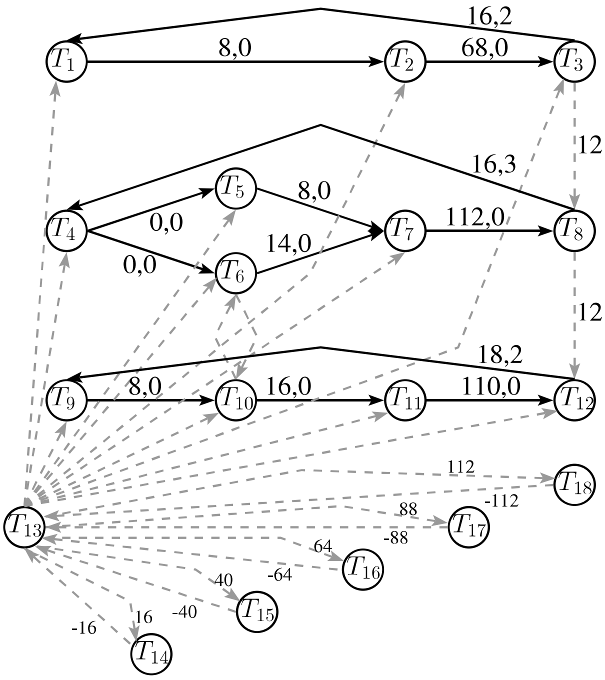

Fig. 3 An example of the lacquer production

Notice that this sequence (modeled by core arcs from T_3 to T_8 and from T_8 to T_{12}), executed each week, is then finished by a general cleanup T_{18} executed at the weekend. In contrast to uniform precedence relations, which represent the flow of the product (vessel with lacquer) of the same iteration, this sequence is independent of the iteration, and therefore, it is modeled by core precedence constraints.

- Dispersing operations are performed on the dispersing line, which is started just once a week. The dispersing line accommodates these operations regardless of their iteration number and, therefore, we model this synchronization by two core arcs with zero length (from T_6 to T_{10} and from T_{10} to T_6).
- Production tasks cannot be scheduled during the night or the weekend, and they cannot be interrupted by the night or the weekend. This constraint is modeled by the dummy task T_{13}, used to model the start of the week, by the fixed-position of tasks $T_{14}, T_{15}, T_{16}, T_{17}$ blocking all resources during the night and by T_{18} blocking all resources during the weekend.

The Gantt chart of a feasible solution of this lacquer problem is shown in Figure 4.

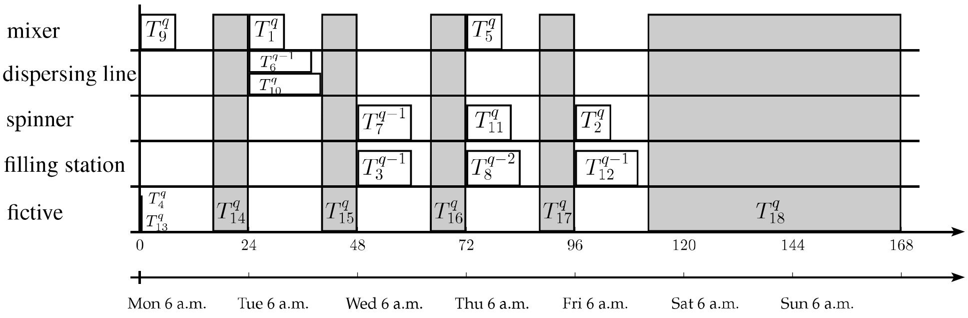

Fig. 4 A feasible solution of the lacquer problem

2.4 Feasibility of Core constraints

If the problem without core precedences is feasible, then there is not always a feasible periodic schedule as shown by the following example, which assumes no resource constraints, only a nonnegative core and uniform precedences:

- G_U is a simple circuit of 3 unit processing time tasks T_1, T_2, T_3, with $l_{1,2} = 1, h_{1,2} = 0$, $l_{2,3} = 1, h_{2,3} = 0$, and $l_{3,1} = 1, h_{3,1} = 1$
- G_C is composed of two arcs: (T_2, T_1) with $c_{2,1} = 1$ and (T_1, T_3) with $c_{1,3} = 1$.

Clearly, in any feasible core, $\sigma_2 < \sigma_1 < \sigma_3$. According to inequality 1, considering a uniform arc (T_i, T_j): $\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i)$.

This will imply that

$$\alpha_j - \alpha_i \geq -h_{ij} + \left\lceil \frac{l_{ij} + \sigma_i - \sigma_j}{\lambda} \right\rceil$$

Applying this inequality to arc (T_1, T_2) leads to $\alpha_2 \geq \alpha_1$ since $\sigma_1 - \sigma_2 < 0$.

Similarly, for arc (T_2, T_3) we get $\alpha_3 \geq \alpha_2$. Consider now arc (T_3, T_1): $\alpha_1 - \alpha_3 \geq -1 + \lceil \frac{1+\sigma_3-\sigma_1}{\lambda} \rceil$. As $\sigma_3 > \sigma_1$, $\lceil \frac{1+\sigma_3-\sigma_1}{\lambda} \rceil \geq 1$, so that $\alpha_1 \geq \alpha_3$ and then all α_i are equal.

But then the uniform arc (T_1, T_2) and the core arc (T_2, T_1) are not compatible.

This result leads us to define the following problems:

- **UCF**: uniform and core feasibility. Given a uniform graph G_U and a core graph G_C, does a feasible periodic schedule that satisfies both exist?
- **UCF_fixed_period**. Given a uniform graph G_U and a core graph G_C, does a feasible periodic schedule with a given period λ that satisfies both exist?
- **RCUCF**: resource constrained uniform and core feasibility. Given a cyclic resource constrained scheduling problem with core constraints, does a feasible periodic schedule exist?
- **RCUCF_fixed_period**: Given a cyclic resource constrained scheduling problem with core constraints, does a feasible periodic schedule with a given period λ exist?

Notice that as finding a periodic schedule with a given period λ (or minimizing) given a cyclic RCPSP problem setting is an NP-hard problem, the problem RCUCF_fixed_period is still NP-hard when core precedence constraints are added.

3 Models for UCF and RCUCF_fixed_period

In this section, we present some previous results of the literature and we show how they can be used to model two of the feasibility problems introduced in the previous section. If we do not consider resource constraints, the feasibility checking problem is expressed as a graph problem, while an ILP is defined for checking the feasibility within a given period in the presence of RCPSP resource constraints.

3.1 State of the art

Most of the results on cyclic scheduling problems aim to minimize the period of a feasible schedule or to build a feasible schedule for a fixed period. The related literature also refers to "modulo scheduling" [26] and "software pipelining" approaches [1], [21]. It is well known that if no resource constraints are considered, given a uniform graph G_U, the feasibility problem, as well as the period minimization problem, can be solved in polynomial time, even if arbitrary integer lengths are considered. According to [27]:

Lemma 1 *Let $H(c)$ stands for the sum of the heights of the arcs in circuit c and $L(c)$ for the sum of the lengths of its arcs. If G_U is feasible, then any circuit c satisfies $H(c) > 0$.*

If this condition is met, we can define a lower bound on the period of any periodic schedule:

$$B = \max_{c \ circuit \ of \ G_U} \frac{L(c)}{H(c)} \tag{3}$$

Following [27] and [24], this bound can be computed in polynomial time, for example with parametric algorithms given in [22], or other algorithms experimented in [8] and we get:

Lemma 2 *if G_U is feasible then $\forall \lambda \geq B$ there exists a periodic schedule with period λ satisfying the constraints of G_U, that can be found in polynomial time.*

Now, finding a modulo schedule of minimal period λ is known to be $\mathcal{NP}$-hard when resources are limited [14].

On the other hand, a bound due to resource constraints only can be defined by

$$\lambda^{res} = \max_{1 \leq k \leq m} \frac{\sum_{T_i \in \mathcal{T}} p_i r_{ik}}{R_k}.$$

That is the minimum such that the renewable resources are not overloaded and it can be easily proven that $\lambda^{opt} \geq \lambda^{res}$.

Several approaches have been investigated in order to solve the cyclic RCPSP problem. Integer Linear Programming [2], greedy iterative heuristics [21][19][26][23][28], mixed approaches [11][10][3], constraint programming [6], and the so called decomposed software pipelining approach [5][7][29][13] from which we borrow some concepts, which will be useful to reformulate the UCF problem in terms of graphs.

Assume that we get a cyclic RCPSP instance, with a uniform graph G_U, and let $(s_j)_{T_j \in \mathcal{T}}$ be a periodic schedule with core $(\sigma_j)_{T_j \in \mathcal{T}}$ and retiming $(\alpha_j)_{T_j \in \mathcal{T}}$.

A retiming is called "feasible" if :

$$\forall (T_i, T_j) \in E_U, \quad h_{ij} + \alpha_j - \alpha_i \geq 0 \tag{4}$$

It can be proven ([7]) that any feasible schedule induces a feasible retiming.

From a feasible retiming α the core graph G_U^α can be defined by removing the arcs from G_U for which $h_{ij} + \alpha_j - \alpha_i > 0$, keeping their length, and adding two dummy nodes T_{start} source of the graph, and T_{end} sink of the graph with zero processing time, with appropriate lengths of their incident arcs.

It has been proved in [5] that for any feasible retiming α, if a core schedule σ is built according to the RCPSP resource constraints and the generalized precedence constraints modeled by the core graph G_U^α then a periodic schedule s with period $C_{max}(\sigma)$) whose retiming is α and whose core schedule is σ can be defined.

This property is used for parallel processors in [13][7] and for RCPSP constraints and precedence delays in [5] to build schedules according to the following decomposed software pipelining algorithm:

1. Find a feasible retiming $(\alpha_i)_{i \in \mathcal{T}}$
2. Build with a list algorithm, a schedule σ of the acyclic graph G_U^α that fulfills the resource constraints
3. Build the periodic schedule whose period is the makespan of σ, whose core is σ and whose retiming is α.

It has been proved in [5] that if:

1. unitary resource demands are assumed ($r_{ik} \in \{0, 1\}$);
2. a retiming minimizing the length of the longest path of G_U^α or based on a schedule build by assuming an infinite amount of resources is used at step 1;
3. a list schedule is used in step 2;

then DSP provides a polynomial time approximation algorithm with a bounded worst case ratio. So, as core precedences only concerns the core schedule construction step, decomposed software pipelining seems to be a convenient approach to solve problems with core precedences. But, as shown in the previous subsection, core precedences may induce infeasibility, which is not handled by this approach. However, we show in next subsection that decomposed software pipelining can be used to formulate UCF as a graph problem.

3.2 A graph model for UCF

In this subsection we do not assume any resource constraints. Assume that a feasible retiming α is given following Equation 4. Recall that any schedule σ of the generalized precedence graph G_U^α with given α induces a periodic schedule satisfiying the constraints of G_U. Now core constraints can be easily added to G_U^α. So let us define the graph $\mathcal{H}^\alpha$ that combines the arcs of G_U^α and the arcs of G_C.

The existence of a periodic schedule with retiming α that fulfills the precedence and core constraints can thus be solved by checking if $\mathcal{H}^\alpha$ contains a cycle.

Notice that once a retiming is given this can be checked in polynomial time.

So, we get the following lemma

Lemma 3 *The UCF problem can be reformulated as follows: Find a feasible retiming α such that the graph $\mathcal{H}^\alpha$ is acyclic.*

We shall use this formulation in next section to study the complexity of this problem.

3.3 An ILP model for RCUCF_fixed_period

Several ILP models have been proposed for cyclic RCPSP problems [9]. According to the authors of [2], the most efficient model is the generalization of time-indexed Eichenberger model. Notice that this model as well as the other model proposed in [16] can be easily extended to handle core precedence constraints and arbitrary lengths and heights.

Eichenberger and Davidson [12] initially describe a model with unit processing time tasks for the feasibility of a periodic schedule with period λ. This model can be extended to handle integer processing time [9]. It uses, as variables, core variables σ_i, integer retiming variables α_i and some time-indexed binary variables related to the core:

y_i^τ such that, $y_i^\tau = 1$ if and only if $\sigma_i = \tau$

$$\sigma_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau \tag{5}$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \quad \forall i \in \{1, \dots, n\} \tag{6}$$

$$\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i) \quad \forall (T_i, T_j) \in E_U \tag{7}$$

$$\sum_{i=1}^{n} \sum_{\max(0, \tau\gamma - p_i + 1)}^{\gamma} y_i^\tau r_{ik} \leq R_k, \quad \forall k \in \{1, \dots, m\}, \forall \gamma \in \{0, \dots, \lambda - 1\} \tag{8}$$

$$y_i^\tau \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, \forall \tau \in \{0, \dots, \lambda - 1\} \tag{9}$$

$$\alpha_i \in \{0, \dots, K - 1\} \quad \forall i \in \{1, \dots, n\} \tag{10}$$

Constraints (7) are the precedence constraints. Constraints (6) state that each generic task has to be started exactly once in the period or, equivalently, that the remainder of the division of σ_i by λ lies in $\{0, \dots, \lambda - 1\}$. With this decomposition, the set of operations such that $y_i^\tau = 1$ directly gives resource constraints (8).

Core precedence constraints can also be included in the model, following Equation (2)

$$\sigma_j - \sigma_i \geq c_{ij} \quad \forall (T_i, T_j) \in E_C$$

So, RCUCF_fixed_period as well as UCF_fixed_period can be modeled as an ILP. Notice that if we try to solve RCUCF, constraint (7) is not linear anymore, since period λ multiplies the retiming variable. So to solve this problem, a binary search on λ should be made, while solving an ILP for each fixed λ.

4 The complexity of UCF

UCF is a key problem, and if we could get solutions of UCF, the resolution of all problems with resource constraints could be made easier using a similar approach as the decomposed software pipelining. However, we prove in this section that whereas it has no resource constraints, the UCF problem is still $\mathcal{NP}$-complete, even if unit processing time tasks and usual precedences (uniform arcs of length 1) are assumed.

4.1 Problem expression with $\mathcal{H}^\alpha$

Let us consider two graphs G_U and G_C, and let us assume that the lengths of the arcs are unitary.

From Lemma 3 the feasibility problem without resource constraints can be expressed as follows:

- Does a feasible retiming α exist such that $\mathcal{H}^\alpha$ is acyclic?

Remark 2 Let α be a feasible retiming.

Then for any circuit c of G_U, there are at most $H(c)$ arcs (T_i, T_j) of c such that $h_{ij} + \alpha(j) - \alpha(i) > 0$. Thus, at most $H(c)$ arcs of c have been removed while defining G_U^α from G_U.

We will use this remark to formulate a proof of NP-completeness of UCF.

4.2 Proof of NP-completeness

Let us consider an instance of the $3-SAT$ problem. We are given n binary variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. Each clause is defined by 3 literals. A literal is either an x_i or its complement $\bar{x_i}$.

The decision problem $3 - SAT$ can be formulated as follows:

- Do binary values of the variables exist such that in each clause, at least one literal is true?

This problem can be illustrated by the following example with 4 variables and 3 clauses:

$$C_1 = \{x_1, \bar{x_2}, x_4\}, \quad C_2 = \{x_2, \bar{x_3}, x_4\}, \quad C_3 = \{\bar{x_1}, \bar{x_3}, \bar{x_4}\}$$

For this example, $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$ provides satisfiability of this system.

Let us associate an instance of our feasibility problem with a $3-SAT$ instance. Let us define, for each variable x_i, two tasks denoted by x_i and $\bar{x_i}$ with an arc of G_U from x_i to $\bar{x_i}$ with length 1 and height 0, and an arc from $\bar{x_i}$ to x_i with length 1 and height 1. These arcs are called “variable arcs”.

Now let us define for each clause C_j four tasks $c_j^0, c_j^1, c_j^2, c_j^3$. Let us add, in G_U, the following arcs with length 1 representing the literals : $(c_j^0, c_j^1), (c_j^1, c_j^2), (c_j^2, c_j^3)$ with height 0 and (c_j^3, c_j^0) with height 2. These arcs are called “clause arcs”.

Notice that according to Remark 2, in any feasible retiming α, exactly one of the variable arcs related to x_i will be removed to get G_U^α, while at most two of the clause arcs related to clause C_j will be removed to get G_U^α.

Let us assume without loss of generality that the literals are ordered with respect to the index number of the associated variables (so if x_i or $\bar{x_i}$ is the first literal (resp. second), x_j or $\bar{x_j}$ the second (resp. third) literal, then $i < j$). Let us now define the core arcs as follows: assume that x_i is the literal $k \in \{1, 2, 3\}$ of clause C_j. Then (x_i, c_j^{k-1}) and $(c_j^k, \bar{x_i})$ are arcs of G_C. If $\bar{x_i}$ is the literal k in clause C_j, then $(\bar{x_i}, c_j^{k-1})$ and (c_j^k, x_i) are arcs of G_C.

Notice that c_j^0 has only one incoming core arc and no outgoing core arc, whereas c_j^3 has only one outgoing core arc and no incoming core arc. c_j^1, c_j^2 both have one incoming and one outgoing core arc. Figure 5 shows the graph associated with the example.

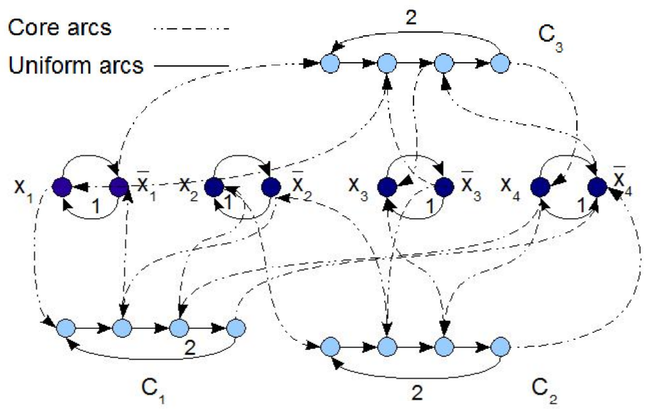

Fig. 5 A polynomial reduction

Lemma 4 *If there is a feasible retiming α such that $\mathcal{H}^\alpha$ is acyclic, then the $3 - SAT$ instance is satisfiable.*

Proof. For each variable x_i, let us set $x_i = 1$ if the arc $(x_i, \bar{x_i})$ is in $\mathcal{H}^\alpha$, $x_i = 0$ otherwise.

We know that for each clause C_j there is at least one arc (c_j^{k-1}, c_j^k) which remains in $\mathcal{H}^\alpha$.

Let us assume that the k^{th} literal of clause C_j is x_i. Then, as we have the core arc (x_i, c_j^{k-1}), the clause arc (c_j^{k-1}, c_j^k) and the core arc $(c_j^k, \bar{x_i})$, if $\mathcal{H}^\alpha$ has no circuit, then the variable arc $(\bar{x_i}, x_i)$ cannot belong to $\mathcal{H}^\alpha$. Therefore, $(x_i, \bar{x_i})$ is in $\mathcal{H}^\alpha$, and $x_i = 1$, so that clause C_j is true.

Let us now assume that the k^{th} literal of clause C_j is $\bar{x_i}$. Then as we have the core arc $(\bar{x_i}, c_j^{k-1})$, the remaining arc (c_j^{k-1}, c_j^k) and the core arc (c_j^k, x_i), the variable arc $(x_i, \bar{x_i})$ cannot belong to $\mathcal{H}^\alpha$, and therefore, $(\bar{x_i}, x_i)$ is in $\mathcal{H}^\alpha$ and $x_i = 0$, so that the clause C_j is true.

$\square$

Let us now consider the reciprocal of this lemma:

Lemma 5 *If the* $3-SAT$ *instance is satisfiable then a retiming* α *such that* $\mathcal{H}^\alpha$ *is acyclic exists.*

Proof. Assume that in the satisfiable assignment $x_i = 1$. Then we set $\alpha_{x_i} = \alpha_{\bar{x_i}} = 0$, so that the variable arc $(x_i, \bar{x_i})$ remains in $\mathcal{H}^\alpha$ whereas the other vanishes. If $x_i = 0$, then we set $\alpha_{x_i} = 0, \alpha_{\bar{x_i}} = 1$, so that the variable arc $(\bar{x_i}, x_i)$ remains in $\mathcal{H}^\alpha$.

Consider now a clause C_j, and its first true literal $k \in \{1,2,3\}$. We define the retiming of tasks of clause C_j as follows:

- If $k = 1$ then $\alpha_{c_j^0} = \alpha_{c_j^1} = 0, \alpha_{c_j^2} = 1, \alpha_{c_j^3} = 2$, so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^0, c_j^1) and (c_j^3, c_j^0).
- if $k = 2$ then $\alpha_{c_j^0} = 0, \alpha_{c_j^1} = \alpha_{c_j^2} = 1, \alpha_{c_j^3} = 2$, so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^1, c_j^2) and (c_j^3, c_j^0).
- if $k = 3$ then $\alpha_{c_j^0} = 0, \alpha_{c_j^1} = 1, \alpha_{c_j^2} = \alpha_{c_j^3} = 2$ so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^2, c_j^3) and (c_j^3, c_j^0).

Figure 6 shows the graph $\mathcal{H}^\alpha$ for the assignment $x_1 = 1, x_2 = x_3 = x_4 = 0$.

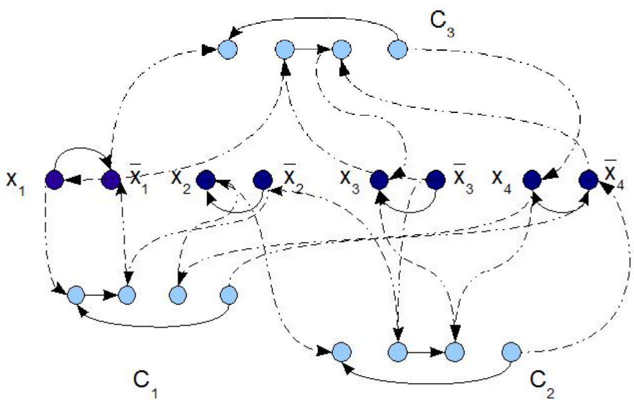

Fig. 6 A graph $\mathcal{H}^\alpha$

Let us assume that there is a circuit μ in $\mathcal{H}^\alpha$.

We first prove that there cannot be a clause C_j for which the arc (c_j^3, c_j^0) belongs to μ. Indeed, the only possible incoming arc to c_j^3 is the arc (c_j^2, c_j^3). The only possible outgoing arc from c_j^0 is (c_j^0, c_j^1). As at least one of them is not in $\mathcal{H}^\alpha$, this arc cannot belong to the circuit.

Let us now prove that if (c_j^{k-1}, c_j^k) belongs to μ then there is another circuit which does not pass through this arc. Notice that c_j^{k-1} has only one incoming arc that may belong to μ (from a variable node x_i (resp.$\bar{x_i}$)) and c_j^k has only one outcoming arc that may belong to μ (to the complemented variable node $\bar{x_i}$ (resp.x_i)), we can define the part μ' of the circuit from $\bar{x_i}$ (resp.x_i) to x_i (resp.$\bar{x_i}$).

By construction, the corresponding literal is true in the satisfiability problem, so the remaining arc $(x_i, \bar{x_i})$ (resp. $(\bar{x_i}, x_i)$) added to μ' defines a new circuit. Hence, if there is a circuit in $\mathcal{H}^\alpha$, there is a circuit that does not use any of the clause arcs.

Let us now consider the variable node of μ with the maximum index. Without loss of generality, assume that it is node x_i (resp. $\bar{x_i}$). The predecessor of x_i on the circuit should be a clause node c_j^k. The predecessor or c_j^k on μ is then a variable node, say $x_{i'}$ (resp. $\bar{x_{i'}}$). Thus, according to the definition of arcs, $\bar{x_i}$ (resp. x_i) should be the k^{th} literal of clause C_j, whereas $x_{i'}$ (resp. $\bar{x_{i'}}$) should be the $(k+1)^t h$ literal of the same clause. But as we assumed that the literals are ordered by increasing index, $i' > i$, which contradicts the fact that x_i (resp. $\bar{x_i}$) has a maximum index on μ. □

From this we deduce our complexity result:

Theorem 1 *The feasibility of a cyclic problem with core precedences and without resource constraint is $\mathcal{NP}-$ complete.*

Proof. Given a retiming α, a simple depth first search algorithm can be used to check the existence of a circuit in $\mathcal{H}^\alpha$ in polynomial time, so the feasibility problem is in $\mathcal{NP}$. Using the previous transformation, and Lemma 4 and 5 we get our result. □

5 Conclusion

In this paper, we defined, in the cyclic RCPSP framework, a new kind of constraints called core precedence constraints, and we illustrated the concept through an industrial example. Then we discussed the interest of the DSP approach to find feasible solutions or to minimize the period. And finally we used this approach to prove that even without resource constraints, introducing core precedence constraints makes the feasibility of a schedule a $\mathcal{NP}$-complete problem. This paper is a first insight for this new kind of constraint. New ideas and algorithms should be further investigated in order to efficiently solve problems with and without resource constraints.

6 Acknowledgments

This work was supported by the Grant Agency of the Czech Republic under the project GACR P103/12/1994.

References

1. Allan, V.H., Jones, R.B., Lee, R.M., Allan, S.J.: Software pipelining. ACM Comput. Surv. **27**(3), 367–432 (1995)
2. Ayala, M., Artigues, C.: On integer linear programming formulations for the resource-constrained modulo scheduling problem. LAAS report 10393. (2010)
3. Ayala, M., Benabid, A., Artigues, C., Hanen, C.: The resource-constrained modulo scheduling problem: An experimental study. Comput. Optim. Appl. **54**(3), 645–673 (2013). DOI 10.1007/s10589-012-9499-2. URL http://dx.doi.org/10.1007/s10589-012-9499-2

4. Behrmann, G., Brinksma, E., Hendriks, M., Mader, A.: Production scheduling by reachability analysis - a case study. In: Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), p. 140.1. IEEE Computer Society Press (2005)
5. Benabid, A., Hanen, C.: Worst case analysis of decomposed software pipelining for cyclic unitary rcpsp with precedence delays. Journal of Scheduling **14**(5), 511–522 (2011)
6. Bonfietti, A., Lombardi, M., Benini, L., Milano, M.: A constraint based approach to cyclic rcpsp. In: CP'11, pp. 130–144 (2011)
7. Calland, P.Y., Darte, A., Robert, Y.: Circuit retiming applied to decomposed software pipelining. IEEE Trans. Parallel Distrib. Syst. **9**(1), 24–35 (1998). DOI http://dx.doi.org/10.1109/71.655240
8. Dasdan, A., Irani, S., Gupta, R.K.: Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In: Design Automation Conference, pp. 37–42 (1999)
9. Dupont de Dinechin, B., Artigues, C., Azem, S.: Resource constrained modulo scheduling. In: C. Artigues, S. Demassey, E. Neron (eds.) Resource-Constrained Project Scheduling: models, algorithms, extensions and applications, Control systems, robotics and manufacturing series, pp. 267–277. ISTE and John Wiley, London (2008)
10. Dupont de Dinechin, B.: From machine scheduling to vliw instruction scheduling. ST Journal of Research **1**(2) (2004)
11. Dupont de Dinechin, B.: Time-indexed formulations and a large neighborhood search for the resource-constrained modulo scheduling problem. In: P. Baptiste, G. Kendall, A. Munier-Kordon, F. Sourd (eds.) 3rd Multidisciplinary International Scheduling conference: Theory and Applications (2007)
12. Eichenberger, A., Davidson, E.: Efficient formulation for optimal modulo schedulers. SIGPLAN - PLDI'97 (1997)
13. Gasperoni, F., Schwiegelshohn, U.: Generating close to optimum loop schedules on parallel processors. Parallel Processing Letters **4**, 391–403 (1994)
14. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: An overview. In: P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (eds.) Scheduling theory and its applications. J. Wiley and sons (1994)
15. Hanzalek, Z., Burget, P., Sucha, P.: Profinet io irt message scheduling with temporal constraints. Industrial Informatics, IEEE Transactions on **6**(3), 369–380 (2010). DOI 10.1109/TII.2010.2052819
16. Hanzalek, Z., Jurcik, P.: Energy efficient scheduling for cluster-tree wireless sensor networks with time-bounded data flows: Application to ieee 802.15.4/zigbee. Industrial Informatics, IEEE Transactions on **6**(3), 438 –450 (2010). DOI 10.1109/TII.2010.2050144
17. Hanzalek, Z., Pacha, T.: Use of the fieldbus systems in academic setting. In: Real-Time Systems Education III, 1998. Proceedings, pp. 93–97 (1998). DOI 10.1109/RTSE.1998.766518
18. Herroelen, W., Leus, R.: Robust and reactive project scheduling: A review and classification of procedures. International Journal of Production Research **42**(8), 1599–1620 (2004)
19. Huff, R.A.: Lifetime-sensitive modulo scheduling. In: In Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation, pp. 258–267 (1993)
20. Kim, E.S., Glass, C.: Perfect periodic scheduling for three basic cycles. Journal of Scheduling **17**(1), 47–65 (2014). DOI 10.1007/s10951-013-0331-3. URL http://dx.doi.org/10.1007/s10951-013-0331-3
21. Lam, M.: Software pipelining: an effective scheduling technique for vliw machines. SIGPLAN Not. **23**(7), 318–328 (1988). DOI http://doi.acm.org/10.1145/960116.54022
22. Levner, E., Kats, V., de Pablo, D.A.L.: Cyclic scheduling in robotic cells: An extension of basic models in machine scheduling theory. In: E. Levner (ed.) Multiprocessor Scheduling: Theory and Applications, pp. 1–20. I-Tech Education and Publishing, Vienna Austria (2007)
23. Llosa, J.: Swing modulo scheduling: A lifetime-sensitive approach. In: Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, PACT '96, pp. 80–. IEEE Computer Society, Washington, DC, USA (1996). URL http://dl.acm.org/citation.cfm?id=882471.883302
24. Munier-Kordon, A.: A graph-based analysis of the cyclic scheduling problem with time constraints: schedulability and periodicity of the earliest schedule. Journal of Scheduling pp. 1–15 (2010). URL http://dx.doi.org/10.1007/s10951-009-0159-z
25. Proth, J.M., Xie, X.: Modélisation, analyse et optimisation des systèmes à fonctionnement cyclique. Masson (1995)

26. Rau, B.R.: Iterative modulo scheduling: an algorithm for software pipelining loops. In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, pp. 63–74. ACM, New York, NY, USA (1994)
27. Robert, Y., Vivien, F.: Introduction to Scheduling. CRC Press, Inc., Boca Raton, FL, USA (2009)
28. Smelyanskiy, M., Mahlke, S., Davidson, E.: Probabilistic predicate -aware modulo scheduling. In: International symposium on Code generation and optimization: feedback-directed and runtime optimization (2004)
29. Wang, J., Eisenbeis, C., Jourdan, M., Su, B.: Decomposed software pipelining: a new perspective and a new approach. Int. J. Parallel Program. **22**(3), 351–373 (1994). DOI http://dx.doi.org/10.1007/BF02577737