# Survey on Periodic Scheduling for Time-Triggered Hard Real-Time Systems

Anna Minaeva Zdeněk Hanzálek

**Abstract**

This survey covers the basic principles and related works addressing the time-triggered scheduling of periodic tasks with deadlines. The wide range of applications and the increasing complexity of modern real-time systems result in the continually growing interest in this topic. However, the articles in this field appear without systematic notation. To address it, we extend the three-field Graham notation to cover periodic scheduling. Moreover, we formally define three example periodic scheduling problems (PSPs) and provide straightforward implementations of these examples in the Satisfiability Modulo Theories formalism with source codes. Then, we present a summary of the complexity results containing existing polynomially solvable PSPs. We also provide an overview of simple state-of-the-art methods and tricks to solve the PSPs efficiently in terms of time. Next, we survey the existing works on PSP according to the resource environment: scheduling on a single resource, on parallel identical resources, and on dedicated resources. In the survey, we indicate which works propose solution methods for more general PSPs that can be used to solve less general ones. Finally, we present related problems that are not periodic by nature to provide an inspiration for a possible solution for the PSP.

## 1 Introduction

A control loop is a typical example of *a hard real-time application* that must operate periodically within the bounds of a stringent deadline. To ensure the safe behavior of the system, the execution of such an application must be repeated periodically. For decades, periodic scheduling problems (PSPs) are found in a wide range of applications, including automotive [12],[83] (e.g., steer-by-wire), avionics [13],[3] (e.g., aircraft control), consumer electronics (e.g., software-defined radio) [7], maintenance (e.g., periodic machine maintenance service) [118], and production (e.g., economic lot scheduling problem) [39].

The increasing complexity of modern real-time systems and the requirement to decrease their size lead to applications sharing resources (e.g., computation units and network links). For example, an Adaptive Cruise Control is a system for road vehicles that automatically adjusts the vehicle speed to maintain a safe distance from the vehicles ahead. The basic periodic tasks to be processed on Electronic Control Units of a car are sensor processing (e.g., vehicle speed,

distance from the vehicle ahead), control algorithm, and actuation (e.g., adjust the throttle position or activate the brake). In these systems, scheduling is applied to ensure that the deadlines of tasks are satisfied.

The two principal approaches to real-time scheduling are: 1. an *event-triggered (ET) approach* [4], where the scheduling decision is made *at the run time* of the system according to the task priorities or deadlines; and 2. a *time-triggered (TT) approach* [64], in which the schedules that satisfy timing requirements by construction are obtained during the design time of a system (sometimes called pre-run-time). Whereas research in the ET domain focuses on providing a response-time analysis (i.e., to prove that all the tasks are completed before their deadlines), the TT approach synthesizes the feasible schedules by design, making schedulability analysis unnecessary. This synthesis is often an NP-hard problem. Real-time systems comprise not only periodic tasks, but also sporadic tasks with unknown request times. The main advantage of the ET approaches is that they can handle sporadic tasks efficiently. On the other hand, the TT approaches are highly predictable, simplifying the design, verification, and certification. To utilize the strengths of both approaches, some authors [35, 52, 51] and technologies (e.g., FlexRay, IEEE 802.1Qbv) combine them. In this article, we focus on *periodic scheduling problems (PSPs)* from the perspective of the TT approach.

The first surveys on the periodic scheduling of real-time systems by Cheng, Stankovic, and Ramamritham in [19] and by Xu and Parnas in [121] were published in the late 1980s – early 1990s and included both TT and ET works. Over the next 20 years, the embedded real-time systems community has mostly focused on ET scheduling. Kopetz in 2003 [64] and later in 2011 [65] thoroughly addressed the TT approach from the implementation point of view. Many papers have been published that address TT scheduling, going far beyond the results of the real-time community. This brings a need to make a coherent picture of the existing works while collecting results from different communities, e.g., periodic maintenance problem [118]. Whereas [25] have published a broad and thorough survey on the works and results in ET scheduling, the work by [69] is the only modern survey on TT scheduling. However, the authors in [69] left space for incorporating works on periodic scheduling problems (PSPs) into the overall picture by classifying them and providing state-of-the-art solutions. This article fills the space.

Existing works on PSPs often do not mathematically formalize the problems they solve, merging the problem definition and problem solution. This lack of formalism complicates both the search for existing solutions and the identification of new challenging PSPs to be solved. The purpose of this article is to provide a categorization of the existing works and to introduce the notation that can simplify the search across different application domains while extending the traditional three-field Graham notation [43].

The PSP can either be *preemptive or non-preemptive*. In the preemptive version, a task can be stopped and completed later after another task. Furthermore, some PSPs assume task *migration*, i.e., execution of one task on different resources, whereas others do not. In this survey, we cover works on

both preemptive and non-preemptive scheduling and works under a migration policy and without the migration policy. However, since the TT scheduling is mainly adopted to safety-critical systems, most of the existing works assume non-preemptive scheduling under no migration policy.

Solving non-preemptive PSPs is computationally more complex than their preemptive counterparts. Whereas some general preemptive PSPs are pseudo-polynomially solvable [11], a non-preemptive PSP is strongly NP-hard in its general form [54]. However, some non-preemptive PSPs with specific restrictions can be solved polynomially, e.g., a PSP with periods forming a geometric progression. Therefore, in this survey, we list the particular restrictions for which polynomial algorithms exist and provide the least general NP-hard PSPs.

In this article, we provide three educative examples of non-preemptive time-triggered PSPs and extend the three-field notation for periodic scheduling. With these examples, we illustrate the possible parameters and constraints for PSP, which influence the problem complexity. We give straightforward implementations of the PSP formulations and provide them in the Satisfiability Modulo Theories formalism in the source code. The main three contributions of this article are: 1. We present a survey on the complexity results containing polynomially solvable PSP formulations. 2. We give an overview of the simple state-of-the-art methods and tricks to solve PSPs efficiently. 3. We present a survey of the existing works, categorizing them into different PSP classes according to the resource environment type. We also provide a list of the related problems, which are not periodic by nature, to find inspiration for more efficient solutions.

The rest of the article is organized as follows: Section 2 presents PSP formulations and notations. It is followed by Section 3, where the complexity of the PSP and the polynomially solvable cases under the particular assumptions are given. Next, some state-of-the-art methods are explained in Section 4. Finally, the survey of existing works is provided in Section 5, before the conclusions are given in Section 6.

# 2 Problem Description and Notation

This section presents the notation and classification for time-triggered PSPs with hard real-time requirements. Additionally, we provide three educative examples of a PSP with different resource environments: single resource, parallel identical resources, and dedicated resources (i.e., the assignment of tasks to the resources is provided). In these examples, we introduce the most common constraints for the specific resource environments.

For simplicity, in this section, we assume non-preemptive scheduling. The most straightforward, but not the most computationally efficient way to formulate a preemptive PSP is to split it to non-preemptable chunks as undertaken in [21]. Formulations without any assumptions on the number of preemptions can be found in [74] and [71].

In PSPs, we consider a set of periodic tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ to be sched-

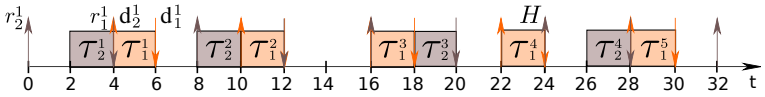



Figure 1: An example schedule with $d_i \leq T_i$ for all $\tau_i \in \mathcal{T}$ on a single resource with a hyper-period $H = 24$ for the set of two tasks $\tau_1$ and $\tau_2$ with periods $T_1 = 6$ and $T_2 = 8$ and processing times $p_1 = p_2 = 2$. The release times are $r_1 = 4$, $r_2 = 0$ and the deadlines are $d_1 = 6$ and $d_2 = 4$. The required maximum jitters are $\overline{jit}_1 = \overline{jit}_2 = 4$. The top-headed arrows of the corresponding color are task release times and the bottom-headed arrows are their deadlines.

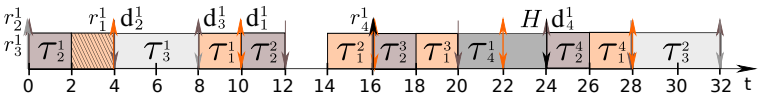



Figure 2: An example schedule with $d_1 > T_1$. Parameters of $\tau_1$ and $\tau_2$ are the same as in Figure 1 except $d_1 = 10$. New tasks are $\tau_3$ and $\tau_4$ with $T_3 = T_4 = 24$, $p_3 = p_4 = 4$, $r_3 = 0$, $r_4 = 16$, $d_3 = 8$, and $d_4 = 24$. Note that the time interval $(2+f\cdot H, 4+f\cdot H)$ for $f = 1, 2, \cdots$ is reserved for a job of task $\tau_1$ from the previous hyper-period. The hatched rectangle represents a projection of this interval to the first hyper-period.

uled during the design time of a system, i.e., before the system run. Periodic task $\tau_i$ is a countably infinite set of iterations called *jobs*, denoted as $\tau_i^k$. A job $\tau_i^k$ is the $k$-th iteration of task $\tau_i$. Jobs of one task have the same inter-arrival *period* $T_i$, the worst-case *processing time* $p_i$, *release time* $r_i$, and *deadline* $d_i$, with the last two related to the start of the period. We assume that the tasks are arranged according to their periods so that $T_1 \leq T_2 \leq \cdots \leq T_n$. Note that all the task parameters are non-negative integers, i.e., they are multiples of an elementary time unit due to the time discretization in TT scheduling [64]. Although there are infinitely many jobs of each task to be scheduled, it is enough to construct a schedule of finite length and repeat it during the system run [20].

Thus, the goal of the considered problem is to find a schedule for the *hyper-period* (schedule length) $H = lcm(T_1, \cdots, T_n)$ with $lcm$ being the least common multiple function. The schedule is defined by the integer start times $s_i^k$ of each task $\tau_i \in \mathcal{T}$ and job $k \in \mathbb{N}^*$. In this work, we denote as $\mathbb{N}^*$ the set of natural numbers without zero and as $\mathbb{N}_0$ the set of natural numbers with zero. Note that the number of jobs of $\tau_i$ in one hyper-period $n_i$ is computed as $n_i = \frac{H}{T_i}$. For the example in Figure 1, there are two periodic tasks with periods $T_1 = 6$ and $T_2 = 8$, and the hyper-period is $H = lcm(6, 8) = 24$. Therefore, the numbers of jobs belonging to the first and second task in the hyper-period are $n_1 = 4$ and $n_2 = 3$, respectively.

*Time-window constraints* state that the earliest start time of a job is its release time $r_i^k$, and the latest completion time of a job is its deadline $d_i^k$, as given by Constraint (1). For the moment, we assume that $d_i \leq T_i$ for each task. The case where $d_i > T_i$ is addressed later in this section (see Figure 2 with

$d_1 = 10$ and $T_1 = 6$, for example).

$$r_i^k = r_i + (k-1)\cdot T_i \leq s_i^k \leq d_i + (k-1)\cdot T_i - p_i = d_i^k - p_i, \quad \tau_i \in \mathcal{T},\ k = 1, \cdots, n_i. \tag{1}$$

Also note that if $d_i \leq T_i$, index $k$ denotes both the period and the iteration of $\tau_i^k$, i.e., $s_i^k \in [(k-1)\cdot T_i, k\cdot T_i]$, whereas if $d_i > T_i$, index $k$ denotes the iteration only and the job can be scheduled later than in the $k$-th period for all jobs of $\tau_1$ in Figure 2.

Moreover, we define the *jitter constraints* for a task as follows. A jitter between two jobs of task $\tau_i$ is the difference between their start times related to the start of their period, as shown in Figure 3. For the case $d_i > T_i$, the jitter is the difference related to the start of its iteration instead, since a job can be in a later period than the start of its iteration, as mentioned earlier. The jitter constraints can be either *absolute or relative*. An absolute jitter of a task is the maximum value among all the job pairs, as given by Constraint (2). For the relative jitter, this constraint is modified to consider the pairs of consecutive jobs only, i.e., $k$ and $k+1$ instead of $k$ and $l$. In this example, the absolute jitter constraint (3) guarantees the maximal possible jitter of task $\tau_i$ given by the value of $\overline{jit}_i$. Note that Constraint (2) also ensures that the maximum jitter is not exceeded for jobs of the same tasks in different hyper-periods due to projection to the start of the job period.

$$jit_i = \max_{\substack{k=1,\cdots,n_i,\\ l=k+1,\cdots,n_i}} (|(s_i^k - (k-1)\cdot T_i) - (s_i^l - (l-1)\cdot T_i)|), \tag{2}$$

$$jit_i \leq \overline{jit}_i,\ \tau_i \in \mathcal{T}. \tag{3}$$

Two special cases regarding the jitter constraints are:

- *zero jitter* with $\overline{jit}_i = 0$, where the start time is fixed in every period. The start time of the $k$-th job of a zero-jitter task $\tau_i \in \mathcal{T}$ is defined by the start time of the first job as $s_i^k = s_i^1 + (k-1)\cdot T_i$ for $k = 2, \cdots, n_i$. These constraints can substitute Constraints (2) and (3) when $\overline{jit}_i = 0$. If all tasks have zero jitter, the problem is also known as strictly periodic scheduling.

- with no assumptions on the task jitter, i.e., without Constraint (3).

For the example in Figure 1, task $\tau_1$ is scheduled with zero jitter, i.e., $jit_1 = 0$, since it is always scheduled at the start of each period ($s_1^1 = s_1^2 - T_1 = s_1^3 - 2\cdot T_1 = s_1^4 - 3\cdot T_1 = 0$), whereas $jit_2 = 2$ as it is scheduled at time 2 in the first and the second periods ($s_2^1 = s_2^2 - T_2 = 2$) and at time 0 in the third period ($s_2^3 - 2\cdot T_2 = 0$), i.e., $jit_2 = |(s_2^1 - (s_2^3 - 2\cdot T_2)| = 2$.

In order to classify various types of periodic scheduling problems, we extend the three-field notation $\alpha|\beta|\gamma$ introduced by [43]. The $\alpha$ field characterizes the resources, the $\beta$ field reflects properties of tasks, and the $\gamma$ field contains the criterion. We list the main elements of the three-field notation for PSPs in

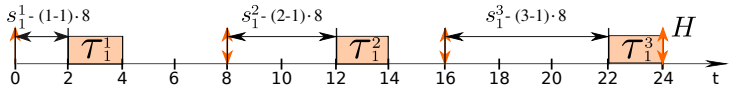



Figure 3: An example schedule with hyper-period $H = 24$ that contains task $\tau_1$ of period $T_1 = 8$, release time $r_1 = 0$, deadline $d_1 = 8$, and processing time $p_1 = 2$. The jitter equals $jit_1 = \max\{|(2-0)-(12-8)|, |(2-0)-(22-16)|, |(12-8)-(22-16)|\} = 4$ according to Equation (2).

Table 2 on page 12. The rows with notations that we newly introduce in this article are colored gray. Note that in this article, we choose the symbol notation consistent with the three-field notation.

In the rest of this section, we introduce some parameters and constraints on three educative examples of non-preemptive PSPs with different resource environments.

## 2.1 Periodic Scheduling Problem on a Single Resource

*Resource constraint* (4) on a single resource guarantee that no two jobs are executed simultaneously. It states that for any pair of jobs of different tasks, we schedule either $\tau_i^k$ before $\tau_j^l$, or $\tau_j^l$ before $\tau_i^k$.


Figure 4: Explanation of the resource constraints

$$\begin{gathered} s_i^k + p_i \leq s_j^l \quad XOR \quad s_j^l + p_j \leq s_i^k, \\ \tau_i, \tau_j \in \mathcal{T} : i < j,\ k = 1, \cdots, n_i,\ l = 1, \cdots, n_j. \end{gathered} \tag{4}$$

Note that in this case XOR is equivalent to OR since both inequalities cannot hold simultaneously.

The described problem on a single resource defined by Constraints (1), (2), (3), and (4) without the criterion function is denoted as $1|T_i, \overline{jit}_i, r_i, d_i \leq T_i|-$. Smaller instances of this problem (with dozens of tasks) can be solved by the Satisfiability Modulo Theory (SMT) solver Microsoft Z3. The SMT models for the problems presented in this paper are available in [82]. Note that if the symbol $r_i$ is omitted in the three-field notation throughout this article, it is set as $r_i = 0$.

An instance of this problem is presented in Figure 1. Note that some problem instances can be infeasible when $d_i \leq T_i$ and feasible when $d_i > T_i$. An example is shown in Figure 2, where new tasks $\tau_3$ and $\tau_4$ are added to tasks $\tau_1$ and $\tau_2$ from Figure 1. With the initial task parameters, i.e., $d_1 = 6$, it is not possible to schedule tasks $\tau_1, \tau_2, \tau_3$, and $\tau_4$. However, an increase in the deadline of $\tau_1$ to $d_1 = 10$ makes it schedulable.

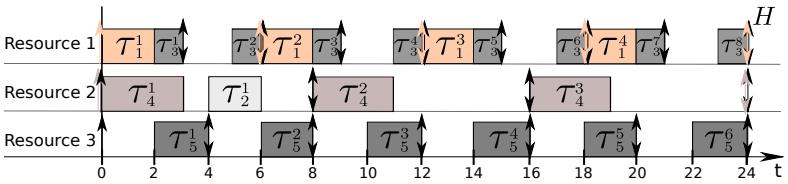



Figure 5: An example schedule for a problem on parallel identical resources with five tasks with periods $T_1 = 6$, $T_2 = 24$, $T_3 = 3$, $T_4 = 8$, $T_5 = 4$, processing times $p_1 = p_2 = p_5 = 2, p_3 = 1, p_4 = 3$, and zero jitter requirement, i.e., $\overline{jit}_i = 0$. The release times and deadlines equal the start and end of the period, respectively, i.e., $r_i = 0$, $d_i = T_i$ for $i = 1, \cdots, 5$.

## 2.2 Periodic Scheduling Problem on Parallel Identical Resources

Parallel identical resources are another typical resource environment. Here, the scheduling algorithm needs to not only find the start times of the tasks, but also the mapping $fix_i : \mathcal{T} \to \mathbb{N}^*$ of each task $\tau_i$ to a resource. For the sake of simplicity, we assume scheduling under a non-migration policy in this example. Note that the formulation can be changed to the migration case by substituting $fix_i$ by $fix_i^k$. For preemptive scheduling, migration can be defined on the job- or task- levels, i.e., whether jobs may migrate to another processor as soon as they are preempted, or chunks of each job must be executed on the same resource, respectively.

The resource constraint (4) for parallel identical resources are formulated only for the pairs of tasks mapped to the same resource as given by Constraint (5).

$$\begin{aligned} &\text{if } fix_i = fix_j \text{ then:} \\ &s_i^k + p_i \le s_j^l \;\; XOR \;\; s_j^l + p_j \le s_i^k, \\ \tau_i, \tau_j \in \mathcal{T} : i < j,\ &k = 1, \cdots, n_i,\ l = 1, \cdots, n_j. \end{aligned} \tag{5}$$

Various optimization criteria are applied in a parallel identical resource environment. Even no criterion considered, i.e., finding a feasible schedule for a given number of resources, is of significant practical importance [87]. The minimization of the number of used resources is a widely assumed criterion as shown in Section 5. We formulate it in Equation (6) by introducing variable $R \in \mathbb{N}^*$ equal to the number of resources. Constraint (7) defines it as the maximum of the mapping variables $fix_i$.

$$\textit{Minimize:} \;\; R \tag{6}$$

$$R = \max_{\tau_i \in \mathcal{T}} fix_i. \tag{7}$$

Thus, a PSP on parallel identical resources denoted as $\mathrm{P}|T_i, \overline{jit}_i, r_i, d_i \le T_i|R$ is defined by Criterion (6) and Constraints (1), (2), (3), (5), and (7). An example

of a schedule for a periodic scheduling problem on parallel identical resources is presented in Figure 5. Here, the SMT implementation accompanying this article [82] shows that the minimum number of resources is 3 for the input data given in the caption of Figure 5.

In some PSPs, the resources to execute certain tasks are limited, i.e., the tasks can be executed only by a subset of resources. This resource environment is called *multipurpose* resources in operational research and scheduling with *arbitrary processor affinities* [44] in real-time community. *Heterogeneous* resources (called unrelated resources in operational research community) are yet another resource environment. Here, the processing time $p_i$ of a task depends on its mapping $fix_i$. Thus, processing time is not a scalar anymore, but it is given by a vector $p_i = \{p_i^1, p_i^2, \cdots, p_i^m\}$, where $m$ is the number of resources. This case can be formulated by substituting $p_i$ by $p_i^{fix_i}$ in Constraints (1) and (5).

## 2.3 Periodic Scheduling of Tasks with Precedence Constraints on Dedicated Resources

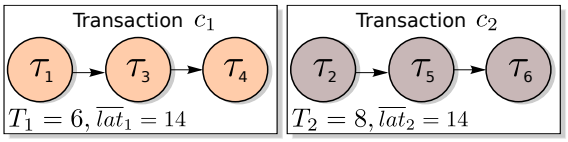
In this example, we consider dedicated resources, therefore, the mapping $\widehat{fix_i}$ is provided for each task $\tau_i \in \mathcal{T}$. In addition, we consider *transaction precedence constraints* that are typically present in periodic scheduling problems with dedicated resources (for example, one control processing unit in a car located close to a camera executes an object detection, the data representing the object parameters are sent via a time-triggered network and subsequently processed by another control processing unit located close to the actuator).

The precedence constraints are defined by the directed acyclic graph $\mathcal{G}$, where each connected component represents one transaction $c_h$ consisting of several tasks connected by the precedence constraints (see Figure 6(a) with two chain transactions $c_1$ and $c_2$). In this section, we assume that all the tasks of the same transaction have the same period, i.e., if $\tau_i \in c_h$ and $\tau_j \in c_h$ then $T_i = T_j$. A more general case with multi-rate dependencies (i.e., those between tasks of different periods) or a cyclic graph of dependencies is addressed in [99], [59], [46], [20], and [111].

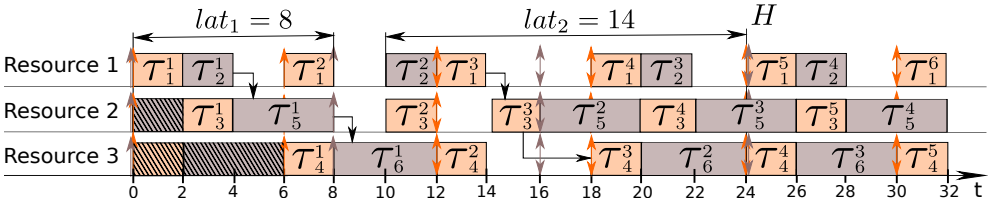
The precedence constraints are formally defined by Constraint (8), where $pred_j \subset \mathcal{T}$ denotes the set of immediate predecessors of $\tau_j$.

$$\begin{aligned} s_i^k + p_i \leq s_j^k, \\ \tau_i, \tau_j \in \mathcal{T} : \tau_i \in pred_j,\ k = 1, \cdots, n_i. \end{aligned} \tag{8}$$

Other constraints typical for periodic scheduling problems with dedicated resources and precedence relations are so called *end-to-end latency constraints*. We define end-to-end latency $lat_h$ of transaction $c_h$ as the maximum time over all the iterations from the start time of the earliest source task (i.e., the task with no predecessor) till the completion time of the latest destination task (i.e., the task with no successor) as given by Constraint (9). The upper bound $\overline{lat_h}$




(a) Precedence relation graph $\mathcal{G}$.




(b) An example schedule with latency $lat_1 = 8$ for $c_1$, and $lat_2 = 9$ for $c_2$. The hatched rectangles represent projections of the executions of tasks $\tau_5^3, \tau_4^4$, and $\tau_6^3$ in the second, third, etc., hyper-periods to the first one.

Figure 6: A periodic scheduling problem on dedicated resources with an example solution. There are two transactions with periods 6 and 8 time units and the end-to-end latency requirements $\overline{lat}_1 = \overline{lat}_2 = 14$, respectively, each comprising three tasks. The task mappings are $\widehat{fix}_1 = \widehat{fix}_2 = 1$, $\widehat{fix}_3 = \widehat{fix}_5 = 2$, $\widehat{fix}_4 = \widehat{fix}_6 = 3$, jitters $\overline{jit}_i = 4$, release times $r_i = 0$, and deadlines $d_i = 2{\cdot}T_i$ for $i = 1, \cdots, 6$.

on the end-to-end latency of each transaction is set in Constraint (10).

$$lat_h = \max_{\substack{\tau_i,\tau_j \in c_h:\ pred_i=\emptyset,\ succ_j=\emptyset,\\ k=1,\cdots,n_i}} (s_j^k + p_j - s_i^k), \tag{9}$$

$$lat_h \le \overline{lat}_h, \qquad c_h \in \mathcal{G}. \tag{10}$$

Depending on the semantics, there can be different ways to define end-to-end latency [34] (also called end-to-end delay). For example, some works define end-to-end latency as an absolute value from the release time of the first source task in a transaction till the completion time of the latest destination task.

The feasible schedule in Figure 6(b) shows an example of two transactions, $c_1$ and $c_2$ shown in Figure 6(a). The end-to-end latency of $c_2$ is given by $lat_2 = \max(s_6^1 + p_6 - s_2^1, s_6^2 + p_6 - s_2^2, s_6^3 + p_6 - s_2^3) = \max(10, 14, 10) = 14$. On the other hand, when the first and the last tasks in a transaction have zero jitter as in transaction $c_1$, the end-to-end latency can be calculated using the first jobs as $lat_1 = (s_4^1 + p_4) - s_1^1 = 6 + 2 - 0 = 8$.

In this example, we consider that the deadlines are not restricted (i.e., that $d_i > T_i$ is possible) to show the formulation of the resource constraint for this more challenging case. This assumption is reasonable due to the precedence constraints, which may enforce some jobs to be scheduled in the subsequent periods for a feasible solution to exist (e.g., tasks $\tau_4$ and $\tau_6$ on Resource 3 in Figure 6(b)). If $d_i > T_i$, jobs of task $\tau_i$ may complete at time $t > H$, i.e., beyond the hyper-period, as $\tau_5^3, \tau_4^4$, and $\tau_6^3$ in Figure 6(b). Since the schedule repeats each hyper-period, the corresponding job is executed at times $t + H$,

$t+2\cdot H$, etc. Therefore, no other task can be scheduled at time $t+H$ on the same resource. To guarantee this, we formulate the resource constraint for the tasks and their replications in further hyper-periods in Constraint (11). Note that the constraints are also created for pairs of different jobs of the same task.

$$\begin{aligned}(s_i^k+f\cdot H)+p_i \leq (s_j^l+h\cdot H) \quad XOR \quad (s_j^l+h\cdot H)+p_j \leq (s_i^k+f\cdot H),\\ f,h \in \{1,2,\cdots,n_H^{max}\},\ n_H^{max} = \left\lceil \frac{\max_{\tau_i\in\mathcal{T},k=1,\cdots,n_i}\ d_i^k}{H} \right\rceil,\\ \tau_i,\tau_j \in \mathcal{T},\ i\leq j : \widehat{fix}_i = \widehat{fix}_j,\ k=1,\cdots,n_i,\ l=1,\cdots,n_j,\\ \text{if } i=j,\ \text{then } k\neq l.\end{aligned} \tag{11}$$

These constraints are needed even for simpler resource environments if $d_i > T_i$ for some task $\tau_i$. For the example of the PSP on a single resource in Figure 2 on page 4, $\tau_1^4$ may collide with $\tau_2^1$ if Constraint (4) is in the model instead of Constraint (11).

The decision version of a PSP on dedicated resources with time windows (1), jitter (2), (3), precedence (8), latency (9), (10), and resource (11) constraints is denoted PD$|T_i$, $\overline{jit}_i$, $r_i$, $d_i$, prec, $\overline{lat}_h|-$. A possible optimization version is the minimization of a weighted sum of latencies that can be performed to improve the behavior of the control system. This problem is denoted as PD$|T_i$, $\overline{jit}_i$, $r_i$, $d_i$, prec, $\overline{lat}_h|\sum w_h\cdot lat_h$ where $w_h$ stands for the weight of transaction $c_h$. SMT formulations solving these two problems (with and without the criterion) can be found in [82].

Finally, a survey-based industry practice study of PSPs in real-time systems can be found in [2].

# 3 Problem Complexity

This section presents an overview of the complexity results for PSPs with different settings. We state the minimal NP-hard (i.e., the least general) and the maximal polynomially solvable (i.e., the most general) PSPs to show the gaps in the complexity results. In the rest of the article, we extensively use the three-field notation to denote the specific PSPs. Table 2 contains the main elements of the three-field notation for the PSP. Whereas most of them have already been introduced in Section 2, the rest will be introduced when required in this section and in Section 5.

## 3.1 NP-hard and NP-complete problems

Addressing the PSPs on a single resource, [54] prove that the non-preemptive PSP with arbitrary task initial phases (i.e., each task is released at $r_i$ and must be finished within $T_i$ time units) and no criterion $1|T_i, r_i, d_i = r_i+T_i|-$ is strongly NP-hard by polynomial reduction from the 3-Partition problem [38]. However, the proof relies on the different release times of the tasks. Furthermore,

Table 1: The list of symbols for the PSP description

| | |
|---|---|
| $H \in \mathbb{N}^*$ | hyper-period (length) of the schedule |
| $\mathcal{T} = \{\tau_1, \cdots, \tau_n\}$ | set of tasks |
| $\tau_i^k$ | $k$-th job of task $\tau_i$ |
| $\mathcal{G} = (\mathcal{T}, E)$ | graph of the precedence relations |
| $c_h \in \mathcal{G}$ | transaction, a connected component of $\mathcal{G}$ |
| $\overline{lat_h} \in \mathbb{N}^*$ | maximum end-to-end latency of $c_h$ |
| $R \in \mathbb{N}^*$ | number of parallel identical resources |
| | |
| *Task attributes* | |
| $T_i \in \mathbb{N}^*$ | period |
| $p_i \in \mathbb{N}^*$ | processing time |
| $\overline{jit_i} \in \mathbb{N}_0$ | maximum jitter |
| $r_i \in \mathbb{N}_0$ | release time |
| $r_i^k \in \mathbb{N}_0$ | job release time |
| $d_i \in \mathbb{N}^*$ | deadline |
| $d_i^k \in \mathbb{N}^*$ | job deadline |
| $pred_i \subset \mathcal{T}$ | set of immediate predecessors |
| $succ_i \subset \mathcal{T}$ | set of immediate successors |
| $fix_i \in \mathbb{N}^*$ | number of the resource to which the task is mapped in the solution schedule |
| $\widehat{fix_i} \in \mathbb{N}^*$ | number of the resource to which the task must be mapped (input parameter) |
| $n_i \in \mathbb{N}^*$ | number of jobs computed as $n_i = \frac{H}{T_i}$ |
| $s_i^k \in \mathbb{N}_0$ | start time of job $k$ of task $\tau_i$ in the schedule. |

Table 2: Main elements of the three-field notation for a PSP (novel colored gray)

| Element | Meaning |
|---|---|
| $1\vert\cdots\vert\cdots$ | one resource |
| $\mathrm{P}\vert\cdots\vert\cdots$ | parallel identical resources as in Section 2.2 |
| $\mathrm{PD}\vert\cdots\vert\cdots$ | dedicated resources as in Section 2.3 |
| $\mathrm{R}\vert\cdots\vert\cdots$ | heterogeneous resources with task processing time dependent on the resource the task is mapped to |
| $\cdots\vert T_i,\cdots\vert\cdots$ | tasks are periodic |
| $\cdots\vert T_i^{harm},\cdots\vert\cdots$ | harmonic periods (a larger period is an integer multiple of a smaller one) |
| $\cdots\vert T_i^{pow2},\cdots\vert\cdots$ | ratio between periods is a power of 2 |
| $\cdots\vert\cdots,r_i,\cdots\vert\cdots$ | release times as in Eq. (1), if omitted set as $r_i = 0$ |
| $\cdots\vert\cdots,d_i,\cdots\vert\cdots$ | deadlines as in Eq. (1) |
| $\cdots\vert\cdots,\overline{jit}_i,\cdots\vert\cdots$ | jitter bounds as in Eq. (3) |
| $\cdots\vert\cdots,\mathrm{prec},\cdots\vert\cdots$ | graph of transaction precedence constraints as in Eq. (8), where tasks of different periods cannot be in the precedence relations |
| $\cdots\vert\cdots,l_{i,j},\cdots\vert\cdots$ | delay $l_{i,j}$ has to be respected whenever a job of task $\tau_j$ starts after a job of task $\tau_i$ |
| $\cdots\vert\cdots,\mathrm{mltrt},\cdots\vert\cdots$ | graph of multi-rate precedence constraints as in Eq. (8), where tasks of different periods can be in the precedence relations |
| $\cdots\vert\cdots,\mathrm{chains},\cdots\vert\cdots$ | precedence constraints in chains |
| $\cdots\vert\cdots,\overline{lat}_h,\cdots\vert\cdots$ | latency bounds as in Eq. (10) |
| $\cdots\vert\cdots,\mathrm{pmtn},\cdots\vert\cdots$ | preemptive scheduling |
| $\cdots\vert\cdots,\mathrm{mlrt},\cdots\vert\cdots$ | migration allowed |
| $\cdots\vert\cdots\vert R$ | minimization of the number of resources |
| $\cdots\vert\cdots\vert\alpha$ | maximization of the scaling factor of the processing times |
| $\cdots\vert\cdots\vert C_{max}$ | minimization of the makespan, i.e., completion time of the last task, with $C_{max} = \max_{\tau_i\in\mathcal{T},k=1,\cdots,n_i}(s_i^k+p_i)$. |
| $\cdots\vert\cdots\vert T_{max}$ | minimization of maximum task tardiness with $T_{max} = \max_{\tau_i\in\mathcal{T},k=1,\cdots,n_i}\{\max(0,s_i^k+p_i-d_i)\}$. |
| $\cdots\vert\cdots\vert\mathrm{ctrl}$ | optimization of control performance |
| $\cdots\vert\cdots\vert n_{pmtn}$ | minimization of the number of preemptions |
| $\cdots\vert\cdots\vert-$ | no criterion to minimize |

the case of the *harmonic period set*, when larger periods are divisible by smaller periods (i.e., for each $T_i \geq T_j$ it holds $T_i \bmod T_j = 0$) seems to be an easier problem, since there are efficient heuristics to solve it ([29]). Nevertheless, [16] strengthened the result of [54] by proving NP-hardness of the PSP with zero release times and harmonic periods. Later, [89] prove that this complexity result holds even if the ratio between the periods is a power of 2, i.e., $T_i = 2^j \cdot T_1$ for $\tau_i \in \mathcal{T}$, $j \in \mathbb{N}_0$ ($T_1$ is the smallest period). Note that for PSPs with no zero-jitter assumptions, "we do not know whether the feasibility problem is in NP" ([16]).

The complexity of the PSP on a single resource with zero jitter requirements $1|T_i, \overline{jit}_i = 0|-$ is addressed by [66]. The authors show that this problem is strongly NP-complete, i.e., it does not admit a pseudo-polynomial time algorithm under standard complexity-theoretic assumptions. Moreover, the same problem with unit processing times $1|T_i, \overline{jit}_i = 0, p_i = 1|-$ is shown to be NP-complete by [7] by the reduction from the graph coloring problem. To strengthen this result, [53] proved that the problem is strongly NP-hard. As a matter of fact, even deciding whether a single task can be added to the set of already scheduled tasks for this PSP is NP-complete, since it is the problem of computing simultaneous incongruences [38]. The summary of all the works addressing the complexity of the PSP is presented in Table 3.

Note that the NP-hardness or NP-completeness of a PSP on a single resource holds for the corresponding PSP in a multiple resources environment (viz., parallel identical, heterogeneous, multipurpose, and dedicated resources). We polynomially transform a single resource PSP to a multiresource PSP by setting the number of resources to 1 in the latter. Thus, the complexity results for PSPs on a single resource presented earlier in this section hold for their multiple resource analogs.

Finally, the only work addressing complexity of a PSP with precedence relations, is [47]. The authors prove that $\text{PD16}|T_i = T, \overline{jit}_i = 0, p_i = 1, \text{chains}, \overline{lat}_h = T|-$, i.e., a monoperiodic PSP on 16 dedicated resources with zero jitter constraints, identical processing times, chain precedence relations, and end-to-end latency constraint equal to chain period, is NP-hard by reduction from the job shop scheduling problem ([10]).

## 3.2 Polynomially Solvable Problems

There exist optimal polynomial algorithms solving PSPs with specific set of task periods. For three different values in the period set, [60] derive an $O(n^4)$ test for the existence of a feasible schedule, and a method of constructing a feasible schedule if one exists. They consider the PSP on a single resource, with zero jitter requirements and unit processing times, $1|T_i \in \{T_1, T_2, T_3\}, d_i = T_i, \overline{jit}_i = 0, p_i = 1|-$. In wireless telecommunications, facilitating three different periodicities increases the power-efficiency of the solutions due to the current practice of scheduling tasks monoperiodically, i.e., with the same period [100].

For parallel identical resources environment, [40] presents an optimal polynomial algorithm to solve the PSP with the set of task periods forming a geo-

Table 3: Works on the complexity of the periodic scheduling problems

| Reference | Problem in the three-field notation | Result |
|---|---|---|
| [54] | $1\|T_i, r_i, d_i = r_i + T_i\|-$ | strongly NP-hard (3-Partition) |
| [16] | $1\|T_i^{harm}, d_i = T_i\|-$ | strongly NP-hard (3-Partition) |
| [89] | $1\|T_i^{pow2}, d_i = T_i\|-$ | strongly NP-hard (3-Partition) |
| [66] | $1\|T_i, \overline{jit}_i = 0\|-$ | strongly NP-complete (3-Partition) |
| [7] | $1\|T_i, \overline{jit}_i = 0, p_i = 1\|-$ | NP-complete (k-coloring) |
| [53] | $1\|T_i, \overline{jit}_i = 0, p_i = 1\|-$ | strongly NP-hard (k-coloring) |
| [47] | PD16$\|T_i = T, \overline{jit}_i = 0, p_i = 1,$ chains$, \overline{lat}_h = T\|-$ | strongly NP-hard (job shop scheduling) |

Table 4: Works on the periodic scheduling problems optimally solvable in polynomial time

| Reference | Problem in the three-field notation | Details |
|---|---|---|
| [60] | $1\|T_i \in \{T_1, T_2, T_3\}, d_i = T_i,$ $\overline{jit}_i = 0, p_i = 1\|-$ | at most three distinct periods |
| [40] | $\mathrm{P}\|T_{i+1} = K \cdot T_i, d_i = T_i\|R$ | $K \in \mathbb{N}^*$, $p_1 + p_i \leq T_1$ for all $\tau_i \in \mathcal{T}$ |
| [27] | $\mathrm{P}\|T_{i+1} = K \cdot T_i, d_i = T_i\|R$ | $K \in \mathbb{N}^*,\ K \geq 3$ |
| [16] | $\mathrm{P}\|T_{i+1} = K \cdot T_i, d_i = T_i\|R$ | $K \in \mathbb{N}^*,\ K = 2$ |
| [66] | $\mathrm{P}\|\{T_i, p_i\}^{harm}, \overline{jit}_i = 0\|R$ | joint harmonic set of periods and processing times |

metric progression, $P|T_{i+1} = K \cdot T_i,\ d_i = T_i|R$. A possible example is a set of four tasks with $T_1 = 3, T_2 = 12, T_3 = 48, T_4 = 192$ for the factor $K = 4$. Unlike the case of harmonic periods, there cannot be tasks with the same period in this case, and the multiplication factor is constant. The algorithm presented by [40] works only when the sum of the processing time of the first task (task with the smallest period) and the processing time of any other task is not more than the smallest period, i.e., $p_1 + p_i \leq T_1$ for all $\tau_i \in \mathcal{T} \backslash \tau_1$. This is a serious limitation, eliminated by [16] and [27] and for $K = 2$ and $K \geq 3$, respectively. Note that these works assume that the deadline equals the period ($d_i = T_i$), which can reduce the flexibility of the solution. Finally, assuming the values of the set of periods and processing times (as a joint set) is harmonic, [66] propose a polynomial time algorithm for a PSP on parallel identical resources with zero jitter requirements, $P|\{T_i, p_i\}^{harm}, \overline{jit}_i = 0|R$. An example of a harmonic period and processing time sets is the case when both are powers of 2, i.e., $T_i = 2^{k_{i_1}}$ and $p_i = 2^{k_{i_2}}$ for some $k_{i_1}, k_{i_2} = 1, \cdots, n_k$. The algorithms in this paragraph are useful when some resource utilization can be sacrificed for a faster solution of the scheduling problem. Moreover, it is sometimes better to form a harmonic period set to reduce the number of required processors [83].

# 4 Basic State-of-the-Art Solutions

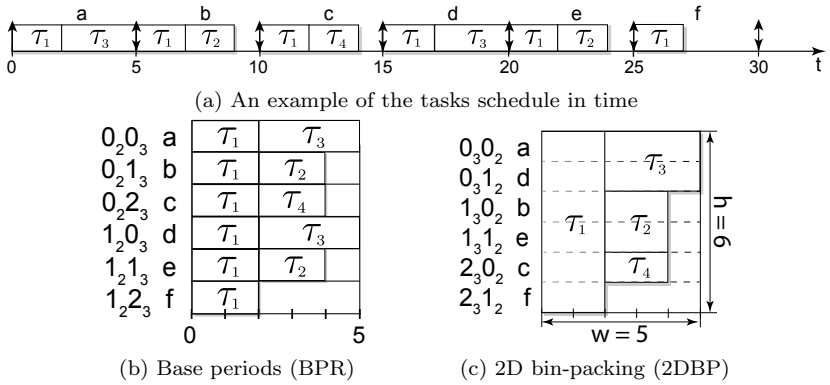
This section presents an overview of the existing techniques to solve PSPs efficiently (in terms of computation time) using specific properties of the problem. First, we present a polynomial transformation of a PSP under specific conditions to a specific version of the 2D bin packing problem also introducing a related data structure to represent periodic schedules. We also present three Integer Linear Programming (ILP) formulations of the PSP on a single resource from Section 2.1 along with their strengths and weaknesses to illustrate the classical solution approach to these problems.

## 4.1 Transformation to Specific 2D Bin-Packing Problem

[73] have shown that the *problem of periodic scheduling on parallel identical resources* $P|T_i^{pow2}, d_i = T_i, \overline{jit}_i = 0|R$ with periods equal to powers of 2 and zero jitter *can be polynomially transformed to a special version of the two-dimensional bin packing (2DBP) problem* with parameters equal to powers of 2, and vice-versa. Later, [45] shows the equivalence of harmonic PSPs and the special version of the 2DBP problem. In this problem, we aim to pack a set of oriented rectangular items into a minimum number of bins with width $w$ and height $h$. The existence of the polynomial transformation gives an insight into the complexity of the mentioned PSP and provides inspiration for its solution. Specifically, heuristics for 2DBP problems show good results in practice (see [29]), which makes it an efficient solution for this type of PSP.

In the rest of this subsection, we describe the transformation. We first introduce a notion of a *base period* $T^{base}$ for the PSP, which is the smallest period of a task set, $T^{base} = T_1$ (task indices are sorted in the non-decreasing order of their respective periods). Then, the schedule for the PSP with harmonic periods can be represented as schedules in base periods placed above each other, as shown in Figure 7(b) for the schedule in Figure 7(a). For simplicity, in Figure 7(b) we denote the schedules in the base periods by letters a, b, $\cdots$, f. To reconstruct the schedule out of the base period representation (BPR) on one row, we put rows of the base periods in the same order going from the top to the bottom.

To transform the schedule in BPR to a 2DBP and vice-versa, we use the mixed-radix numeral system [61] to index the rows. In this system, the numerical base varies from position to position and the corresponding number in a decimal system is computed as the sum of digits multiplied by the product of the previous bases, i.e., $d^1_{b_1} d^2_{b_2} \cdots d^n_{b_n} = d^1 + d^2 \cdot b_1 + d^3 \cdot b_1 \cdot b_2 + \cdots + d^n \cdot \prod_{i=1}^{n-1} b_i = \sum_{k=1}^{n} d^k \cdot \prod_{i=1}^{k-1} b_i$. For example, $2_3 1_2 = 2 + 1 \cdot 3 = 5$. In our case, the bases are the quotients of the division of $(l+1)$-th by $l$-th period in *an ordered set $S$ of unique task periods*. Given a PSP instance defined by a set of tasks with harmonic periods and zero jitter, we transform it to a 2DBP instance in the following way. The width of the bins equals the base period, whereas its height is the number of base periods in the hyper-period, i.e., $w = T^{base}$ and $h = \frac{H}{T^{base}}$. The items are the tasks with the width equal to the processing time and the




(a) An example of the tasks schedule in time

(b) Base periods (BPR)

(c) 2D bin-packing (2DBP)

Figure 7: The transformation of the problem of scheduling tasks with harmonic periods and zero jitter requirements to a specific version of a 2D bin packing problem for the set of 4 tasks with periods $T_1 = 5$, $T_2 = T_3 = 15$ and $T_4 = 30$ and processing times $p_1 = p_2 = p_4 = 2$ and $p_3 = 3$.

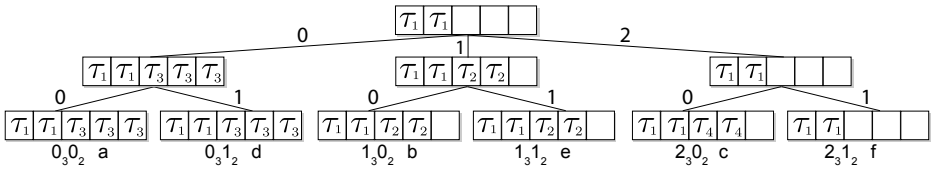



Figure 8: Bin tree representation for the schedule in Figure 7.

height equal to the hyper-period divided by a task period. When we have a solution to the PSP in the BPR, the solution to the 2DBP is obtained by putting the rows in the order of digits in the mixed-radix system with the swapped bases as shown in Figure 7(b). Note that for the solution of the 2DBP problem to be valid for the initial periodic scheduling problem, each item can be placed anywhere on the x-axis, but only on a multiple of its height on the y-axis.

Another possible representation is a *bin tree* [8, 31], which is a data structure often used to derive schedules for periodic zero-jitter tasks with harmonic periods [31, 60, 8, 77]. Bin trees serve to simplify the development of efficient polynomial-time heuristic algorithms. The number of levels of a bin tree is the number of elements in $S$ and the branching factor on level $l$ equals the quotient of the division of $(l+1)$-th by $l$-th period in $S$. Each leaf node of a bin tree is one base period, where only a subset of the tasks is scheduled. Nodes on level $l$ may contain only tasks with periods less or equal to the $l$-th element in $S$. The final schedule can be reconstructed from the leaf nodes placed in the order of increasing number in the mixed-radix numeral system.

## 4.2 Integer Linear Programming Formulations

Due to the NP-hardness of the general PSP, the existing formalisms are often used to obtain the optimal solution (e.g., ILP, SMT, or Constraint Programming (CP)). This is done to estimate the quality of the heuristic solutions compared to the optimal solution on smaller problem instances. Since the models in these formalisms can be formulated, changed, and extended reasonably fast, their usage is especially appropriate during the stage when the model is undergoing changes. Moreover, there are solvers to explore the vast solution space efficiently (e.g., Gurobi Optimizer or IBM ILOG CPLEX for ILP, Microsoft Z3 for SMT, and CP Optimizer for CP).

Whereas the best optimal solvers on modern computing resources can handle PSP problem instances roughly with lower hundreds of tasks (viz., 100-200-300) in a reasonable time (viz., up to an hour of computation), heuristic algorithms can find good enough solutions for problem instances with thousands of tasks. Furthermore, it is possible to improve the scalability of the optimal solvers with decomposition methods preserving the optimality of the solution (e.g., column generation ([107])). However, note that the efficiency of the solution heavily depends on instance characteristics. Both size and complexity of the solution play an important role. Specifically, problem instances with harmonic periods, where the hyper-period equals the largest task period are typically faster and easier to solve than those with non-harmonic periods. The reason is two-fold: 1. a higher hyper-period result in a larger number of task jobs (and, consequently, decision variables and constraints, for the case with no zero jitter requirements) and 2. there are more potential collisions between jobs in the schedule (especially for zero-jitter case). Some other possible reasons for a higher computational complexity of instances are higher utilization (i.e., the sum of task execution times divided by their periods) and large processing times, resulting in a smaller space of feasible solutions.

In this work, we elaborate on the ILP formalism to solve PSPs. We provide the SMT formulation in codes accompanying this article [82]). Additionally, unlike the ILP and SMT approaches, the efficiency of the CP formulation heavily depends on the implementation in the solver (e.g., using a specific solver-defined data structure for scheduling can accelerate the search tens of times) and can be found in [99, 84]. Finally, the efficiency of the ILP may significantly depend on the choice of the decision variables and constraints representation [32], and it is a widely used tool to solve the problems in real-time systems domain [5, 105, 42, 12, 73]. A slightly outdated comparison of ILP, CP, and SMT using solvers CPLEX, Yices2, and MiniZinc/G12, respectively, on different PSPs can be found in [41].

For simplicity and conciseness, in this section we consider the PSP on a single resource given by the time-window constraint (1), jitter constraints (2),(3), and resource constraint (4) defined in Section 2.1. To use the ILP, we need to linearize (i.e., reformulate) the jitter and resource constraint since the absolute value and XOR are not linear operators. The linear version of the jitter constraints is given by Constraint (12), in which we substitute the absolute value operator with two inequality constraints.

$$\begin{aligned} (s_i^k - r_i^k) - (s_i^l - r_i^l) &\leq \overline{jit}_i, \\ (s_i^k - r_i^k) - (s_i^l - r_i^l) &\geq -\overline{jit}_i, \\ \tau_i \in \mathcal{T},\ k = 1, \cdots, n_i - 1,\ l &= k+1, \cdots, n_i. \end{aligned} \tag{12}$$

We provide the formulations of the resource constraint within the three widely used ILP formulations for scheduling problems: 1. the time-indexed (TI) [30] used in [32, 93], 2. the relative-order (RO) [6] (also called disjunctive) used in [3], [32], [97], [12], [42], [125], [105], [83], [81], [15], [74], and 3. the position-based (PB) [72] (also called rank-based or assignment-based) used in [26], [63]. Almost all existing works on PSPs employing ILP used the RO formulation due to its straightforward formulation. However, sometimes the TI and PB formulations may result in a faster solution [62], directly or by a problem decomposition (e.g., for branch-and-price algorithm) [85]. Note that there is the fourth type of ILP formulation for PSPs with harmonic periods and zero-jitter inspired by the transformation to the 2D bin-packing problem presented in the previous subsection. We point the interested reader to [73] and [32].

An extension to the other presented PSPs is mostly straightforward. For example, similarly to the jitter constraints (12), the latency constraint (9) for the PSP with precedence constraints on dedicated resources can be linearized by substituting the max operator by multiple inequalities (e.g., in [84]).

#### 4.2.1 Time-Indexed Formulation

In the TI formulation, the binary decision variables indicate the allocation of a time unit by a task:

$$x_{i,k,t} = \begin{cases} 1, & \text{if job } k \text{ of task } \tau_i \text{ starts executing at time } t. \\ 0, & \text{otherwise.} \end{cases}$$

Constraint (14) ensures the satisfaction of the resource constraint (4). Finally, the start time variables are set by Constraint (15).

$$\sum_{t=0}^{H-1} x_{i,k,t} = 1, \qquad \tau_i \in \mathcal{T},\ k = 1, \cdots, n_i. \tag{13}$$

$$\sum_{i=1}^{n} \sum_{k=1}^{n_i} \sum_{t=v-p_i+1}^{v} x_{i,k,t \bmod H} \leq 1, \qquad v = 0, \cdots, H-1. \tag{14}$$

$$s_i^k = \sum_{t=0}^{H-1} t \cdot x_{i,k,t}, \qquad \tau_i \in \mathcal{T},\ k = 1, \cdots, n_i. \tag{15}$$

In the TI formulation, there are $H \cdot \sum_{\tau_i \in \mathcal{T}} n_i$ binary decision variables and $3 \cdot \sum_{\tau_i \in \mathcal{T}} n_i$ (Constraints (1),(13)) $+ \sum_{\tau_i \in \mathcal{T}} \frac{n_i \cdot (n_i - 1)}{2}$ (Constraint (12)) $+H$ (Constraint (14)) constraints. We do not count the variables $s_i^k$ and Constraint (15), as they can be substituted directly in the jitter and time window constraints. The number of variables grows with the increasing hyper-period $H$, which can be computationally demanding for real-life problems requiring fine time granularity. On the other hand, for coarser granularity and high number of tasks, the TI formulation may show better results than the RO and PB formulations.

### 4.2.2 Relative-Order Formulation

This formulation is based on binary decision variables $y_{i,k,j,l}$ responsible for the order of two jobs on the same resource. They are defined as

$$y_{i,k,j,l} = \begin{cases} 1, & \text{if job } k \text{ of task } \tau_i \text{ finishes before job } l \text{ of task } \tau_j \text{ starts,} \\ 0, & \text{otherwise,} \end{cases}$$

Moreover, the start time variable $s_i^k$ is used with the time window and jitter constraints remaining without changes. The resource constraint (4) are linearized as in the Inequalities (16).

$$\begin{aligned} s_i^k + p_i &\leq s_j^l + \mathbb{M} \cdot (1 - y_{i,k,j,l}), \\ s_j^l + p_j &\leq s_i^k + \mathbb{M} \cdot y_{i,k,j,l}, \\ \tau_i, \tau_j \in \mathcal{T} : i < j,\ k = 1, \cdots, n_i,\ l &= 1, \cdots, n_j, \end{aligned} \tag{16}$$

with $\mathbb{M}$ being a sufficiently big integer positive constant. The intuition behind is that the XOR condition in the resource constraint (4) is substituted for the relative order variables that always turn off one of the two constraints. By regrouping the terms in the Inequality (16), the linearized resource constraint can be presented as in Inequalities (17). We will use this form for a later comparison.

$$p_i - \mathbb{M} \cdot (1 - y_{i,k,j,l}) \leq s_j^l - s_i^k \leq \mathbb{M} \cdot y_{i,k,j,l} - p_j \tag{17}$$

There are in total $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} n_i \cdot n_j + \sum_{i=1}^{n} n_i$ decision variables and $2 \cdot \sum_{i=1}^{n} n_i$ (time-windows constraint) $+ \sum_{i=1}^{n} \frac{n_i \cdot (n_i-1)}{2}$ (jitter constraints) $+2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} n_i \cdot n_j$ (resource constraint) constraints in the formulation. Unlike the TI formulation, where the main computational inefficiency is due to the large number of variables, for the RO formulation, the resource constraint are the reason for the long computation time. The introduction of $\mathbb{M}$ typically results in a higher duality gap [24], reducing the efficiency of the solution process.

**Zero-jitter case** Whereas the resource constraints (16) guarantee no collisions for any pair of jobs, the following formulation can be used for two tasks $\tau_i, \tau_j \in \mathcal{T}$ with zero jitter requirements, i.e., $\overline{jit}_i = \overline{jit}_j = 0$, to improve the computational efficiency of the formulation. Particularly, [68] show that Constraint (18) guarantees the absence of collisions with the $\gcd(T_i, T_j)$ being the greatest common divisor of the two periods, while [97] provide an alternative explanation using the Bezout identity. Note that a schedule of a task $\tau_i$ with zero jitter requirements is fully defined by the start time $s_i^1$ of its first job, since $s_i^{k+1} = s_i^k + T_i$. Moreover, since Constraint (18) allows checking the feasibility of a given solution in polynomial time, PSPs with zero-jitter requirements lie in NP.

$$\begin{aligned} &(s_i^1 - s_j^1) \bmod \gcd(T_i, T_j) \geq p_j, \\ &(s_j^1 - s_i^1) \bmod \gcd(T_i, T_j) \geq p_i, \qquad \tau_i, \tau_j \in \mathcal{T}. \end{aligned} \tag{18}$$

This constraint guarantees a collision-free schedule even when $d_i > T_i$, whereas Constraint (16) requires additional constraints for jobs in further hyper-periods as in Constraint (11) if deadlines can be larger than periods.

We linearize Constraint (18) in Constraint (19), in which we introduce a quotient variable $q_{i,j} \in \mathbb{Z}$ for each ordered pair of tasks $\tau_i, \tau_j \in \mathcal{T}$ as in [97], which is a definition of the modulo operator.

$$\begin{aligned} p_i &\leq s_j^1 - s_i^1 + q_{i,j} \cdot \gcd(T_i, T_j) < \gcd(T_i, T_j), \\ p_j &\leq s_i^1 - s_j^1 + q_{j,i} \cdot \gcd(T_i, T_j) < \gcd(T_i, T_j). \end{aligned} \tag{19}$$

Summing up the two Inequalities (19) results in $p_i + p_j \leq q_{i,j} \cdot \gcd(T_i, T_j) + q_{j,i} \cdot \gcd(T_i, T_j) < 2 \cdot \gcd(T_i, T_j)$. From the right inequality, it follows that $q_{i,j} + q_{j,i} < 2$. From the left inequality, since the quotients are integer variables and without loss of generality we can assume that the tasks have non-zero processing times, it follows that $q_{i,j} + q_{j,i} \geq 1$. This leads to the substitution $q_{j,i} = 1 - q_{i,j}$. Thus, for each pair of zero-jitter tasks, the resource constraint (4) can be substituted by Constraint (20).

$$p_i \leq s_j^1 - s_i^1 + q_{i,j} \cdot \gcd(T_i, T_j) \leq \gcd(T_i, T_j) - p_j, \qquad \tau_i, \tau_j \in \mathcal{T}. \tag{20}$$

Note that this inequality is similar to Inequality (17) with $q_{i,j}$ playing the same role as $y_{i,k,j,l}$ and $\gcd(T_i, T_j)$ - the same as $\mathbb{M}$.

**Schedulability conditions for zero-jitter case** As first proven by [68] (and it follows from Inequlities (18)), a necessary and sufficient condition for scheduling exactly two zero-jitter tasks can be stated as in Inequality (21).

$$p_1 + p_2 \leq \gcd(T_1, T_2). \tag{21}$$

For three and more tasks one has to verify Inequality (22). However, this is the sufficient condition only.

$$\sum_{i=1}^{n} p_i \leq \gcd(T_1, \cdots, T_n). \tag{22}$$

An example of the task set with unsatisfied Inequality (22) but existing schedule on one resource is a task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ with periods $T_1 = 4, T_2 = 6$, and $T_3 = 12$ and processing times $p_1 = p_2 = p_3 = 1$. An example of a feasible schedule is $s_1^1 = 0, s_2^1 = 5$, and $s_3^1 = 1$.

For three tasks to be schedulable, [97] prove the necessary and sufficient condition as in Inequality (23). For more than three tasks, Condition (23) with $\mathbf{g} = \gcd(T_1, \cdots, T_n)$, becomes sufficient only.

$$\sum_{i=1}^{n} p_i \leq \mathbf{g} \cdot \left\lfloor \frac{1}{2} \cdot \frac{\gcd(T_1, T_2) + \gcd(T_2, T_3) + \cdots + \gcd(T_n, T_1)}{\mathbf{g}} \right\rfloor. \tag{23}$$

### 4.2.3 Position-Based Formulation

For this formulation, we introduce binary decision variables reflecting the position of the job in the job permutation on the resource:

$$z_{i,k,f} = \begin{cases} 1, & \text{if job } k \text{ of task } \tau_i \text{ is at the } f\text{-th position on the resource,} \\ 0, & \text{otherwise.} \end{cases}$$

For the example in Figure 1, $z_{2,3,6} = 1$, $z_{2,1,1} = 1$, and $z_{1,1,1} = 0$.

To ensure that each position is occupied by one job and each job occupies one position, Constraints (24) and (25) are introduced, respectively. The number of positions $n_p$ is the number of all task jobs, i.e., $n_p = \sum_{i=1}^{n} n_i$.

$$\sum_{i=1}^{n} \sum_{k=1}^{n_i} z_{i,k,f} = 1, \qquad f = 1, \cdots, n_p. \tag{24}$$

$$\sum_{f=1}^{n_p} z_{i,k,f} = 1, \qquad \tau_i \in \mathcal{T},\ k = 1, \cdots, n_i. \tag{25}$$

The second decision variable is $t_f \in \mathbb{N}_0$, which is the start time of the task job on position $f$ in the schedule. For the example in Figure 1, $t_2 = 4$, $t_5 = 16$, and $t_7 = 22$. Thus, the vector of the $t_f$ variables is a reshaped and reordered matrix of $s_i^k$ variables.

Finally, we introduce parameter variables indexed by $f$ in Constraints (26), where we sum the corresponding parameter multiplied by the $z_{i,k,f}$ variable over all tasks and jobs.

$$\begin{aligned} \overline{r}_f &= \sum_{i=1}^{n}\sum_{k=1}^{n_i} r_i^k \cdot z_{i,k,f}, \qquad \overline{d}_f = \sum_{i=1}^{n}\sum_{k=1}^{n_i} d_i^k \cdot z_{i,k,f}, \\ \overline{p}_f &= \sum_{i=1}^{n}\sum_{k=1}^{n_i} p_i \cdot z_{i,k,f}, \qquad \overline{jit}_f = \sum_{i=1}^{n}\sum_{k=1}^{n_i} jit_i \cdot z_{i,k,f}. \end{aligned} \tag{26}$$

Then, we replace the resource constraint (4) by Constraint (27), whereas the time-window constraint (1) and jitter constraints (12) are reformulated using variables $t_f$ instead of $s_i^k$ with the parameter variables $\overline{r}_f, \overline{d}_f, \overline{p}_f$, and $\overline{jit}_f$, defined by Constraint (26).

$$t_f \geq t_{f-1} + \overline{p}_{f-1}, \qquad f = 2, \cdots, n_p. \tag{27}$$

Note that jitter constraint should be formulated for all pair of jobs, with active only those for jobs of the same task (using big $\mathbb{M}$ trick as in Constraint (16)).

This formulation comprises $\sum_{i=1}^{n} n_i \cdot \sum_{i=1}^{n} n_i + \sum_{i=1}^{n} n_i$ decision variables and $5 \cdot \sum_{i=1}^{n} n_i + \sum_{i=1}^{n} n_i \cdot (\sum_{i=1}^{n} n_i - 1)$ constraints without the variables and constraints in Equations (26) as they can be directly substituted into the other constraints. Note that this model becomes non-linear when the processing times or other parameters defined by Equations (26) are not given (i.e., they are decision variables). Finally, this formulation is non-linear for PSPs with precedence constraints as in Section 2.3. For this case, an event-based formulation [62] is more suitable.

# 5 Survey of Works on Periodic Scheduling Problems

In this section, we first survey the works addressing non-preemptive PSPs. Then, we present works solving preemptive PSPs. Finally, we list problems similar to the PSP and state how they are different from it.

The works in the non-preemptive and preemptive subsections are presented in the order given by the three-field notation of the solved problem. The works are presented first based on the resource environment type, i.e., the first field in the three-field notation. For two works solving PSPs with an identical resource environment, the order is defined by the assumptions on the task periods. The third aspect is the release times, etc., and the last differentiator is the optimization criterion. We present the first works solving more general PSPs, breaking the tie based on the publication date (earlier first). Figure 9 shows the most general PSP in the vertical rectangle and the corresponding less general parameter values in the horizontal rounded rectangles. For example, an algorithm for a PSP with arbitrary periods ($T_i$) can be used to solve the problem with

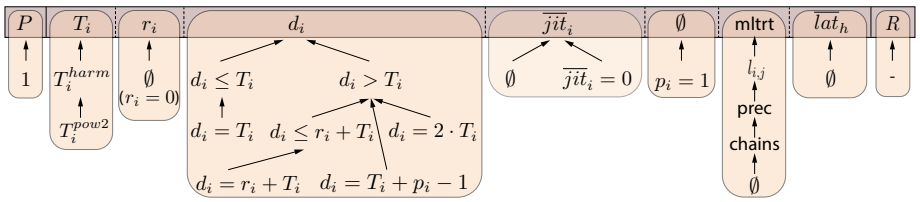



Figure 9: The scheme of PSP reductions in the three-field notation.

harmonic periods ($T_i^{harm}$) or with periods which are powers of 2 ($T_i^{pow2}$). Also, a non-preemptive PSP with identical processing times ($p_i = 1$) is often equivalent to the corresponding preemptive PSP unless there are limitations on the number and locations of preemptions.

Table 2 in Section 2 presents an explanation of the frequently used elements in the three-field notation of the problems considered by the existing works. However, due to the wide application area, some articles solve problems with non-standardized constraints or criteria that we will introduce ad hoc. Note that articles marked with $\star$ in the second field of the three-field notation consider additional application-specific constraints that we do not list to keep the focus. Moreover, we provide the most general version of the addressed problems. For instance, if a work addresses both the case of harmonic and arbitrary periods, we state the latter.

## 5.1 Works on Non-Preemptive PSP

In this subsection, we first present works solving PSPs on a single resource and on parallel identical resources and then works addressing PSPs on dedicated and heterogeneous resources. The resource environments are merged this way due to the conceptual similarity of the works. Whereas works belonging to the former resource environments are mostly theoretical often proposing schedulability conditions, works in the latter ones are often applied, solving more practical scheduling problems.

### 5.1.1 Scheduling on a Single Resource and on Parallel Identical Resources

Tables 5 and 6 present the works in this subsection. The problem of scheduling with zero-jitter requirements (and zero release times) received a lot of attention on both a single resource and on parallel resources. [76] address the PSP on a single resource and restricts the task deadlines to be equal to their periods. The authors provide schedulability conditions for two tasks and for general sets of tasks and propose a heuristic, which is based on these conditions. Solving the same PSP, [77] consider separately the cases of harmonic and non-harmonic periods in the schedulability analysis. For the harmonic case, the necessary and sufficient schedulability condition (28) for distinct task periods is presented.

On the other hand, when two tasks may have the same period (and the periods are harmonic), the authors propose a cumbersome sufficient schedulability condition. For the case of non-harmonic periods, the authors present a local schedulability condition assuming that a set of tasks is already scheduled and a new task is to be scheduled.

$$p_1 + p_i \leq T_1,\ \tau_i \in \mathcal{T} \backslash \tau_1. \tag{28}$$

For the same PSP, [126] propose a constructive heuristic, building a schedule task-by-task in the order of the increasing greatest common divisor of the task pairs. Solving this PSP on parallel identical resources, [18] present a sufficient local schedulability condition for a task $\tau_i$ to be schedulable with a set of already scheduled tasks. The condition states that if the set of possible start times where $\tau_i$ with unit processing times can be scheduled contains $p_i$ consecutive time slots, the task is schedulable. The authors present a First-Fit based heuristic algorithm using this condition. Finally, [56] generalises the results of [77] to parallel identical resources and proposes a heuristic using these generalized results.

A number of works further considered no explicit deadlines in the PSP with zero-jitter requirements. In this case, the task deadlines $d_i$ equal to $T_i + p_i - 1$ due to the repetition of a task at the same time in each period. For this PSP on parallel identical resources, [68] propose an approximation algorithm, which constructively assigns periodic tasks to periodic intervals of the idle resource time. Both cases, with and without the migration policy, are considered. Later, [66] address the same problem under the no migration policy and proposes an approximation algorithm conceptually similar to the approximation algorithm in [68].

For the same PSP on parallel identical resources with minimization of the number of resources, [31] show that a simple First-Fit algorithm is a 2-approximation algorithm for the case of harmonic periods and prove that it cannot be improved, unless $P = NP$. In the First-Fit algorithm, tasks are added in a non-decreasing order of their period lengths using the schedule representation of bin trees (described in Section 4). Additionally, the authors propose an asymptotic polynomial time approximation scheme for the case of a bounded number of distinct harmonic task periods. Finally, the First-Fit inspired heuristic is proposed to solve the problem with general periods. Later, [32] propose multiple Integer Programming formulations based on structural properties to solve this problem with additionally introduced memory and redundancy constraints.

[93] address the PSP with zero jitter, under the migration policy, and assuming identical processing times. The authors prove that partitioning the set of tasks to subsets with relatively prime periods maintains the optimality of the solution using the Chinese Reminder Theorem. Then, they run the proposed ILP model on each subset separately. The efficiency of this decomposition depends on the task periods, since the closer the periods are to being harmonic, the less the subsets are. [123] also address the PSP with unit processing times and propose a schedulability condition based on the reduction of the PSP to

a special case of a graph coloring problem. The graph representation helps to decide faster on the minimal distance between two tasks or on adding a new task.

A possible criterion is the minimum scaling factor $\alpha \geq 1$ among all the processing times so that the resource constraint (29) hold. Its maximization allows for the higher flexibility in the cases when the tasks worst-case processing time is underestimated or changed. Maximizing the minimum scaling factor, [3] consider a PSP on parallel identical resources with zero jitter requirements and additional hardware constraints. The authors propose an ILP model and a game theory-inspired heuristic to solve the problem.

$$s_i^k + \alpha \cdot p_i \leq s_j^l \quad XOR \quad s_j^l + \alpha \cdot p_j \leq s_i^k \tag{29}$$

Furthermore, [97] solve the same problem with additional time lag constraints, where delay $l_{i,j} \geq 0$ has to be respected whenever a job of task $\tau_j$ starts after a job of task $\tau_i$. The authors propose a necessary condition of the task schedulability that is sufficient for, at most, 3 tasks. The authors refine the solution of [3].

The PSP on a single resource with zero jitter and multi-rate precedence constraints is considered by [58] and [22]. The former work limits the multi-rate constraints to the case when a predecessor or a successor task can have a multiple period of the period of its successor or predecessor tasks, respectively, with a multiplier $n$. Then, the predecessor task must be executed $n$ times before the successor starts or vice-versa. The authors present and prove a sufficient schedulability condition for scheduling a set $\mathcal{T}$ of tasks: $\sum_{\tau_i \in \mathcal{T}} p_i \leq \gcd\{T_j, \tau_j \in \mathcal{T}\}$, which is an extension of Inequality (21). However, this condition is too restrictive: for instance, it is possible to construct a schedule for a set of tasks with periods 6, 10, and 15 and unit processing times, where the condition is not satisfied. The authors prove that this condition becomes sufficient for the set of tasks with identical pairwise greatest common divisors of the periods. This work also presents a necessary and sufficient condition for the schedulability of one task with the set of already scheduled tasks based on their greatest common divisors. [55] extends the results of [58] for parallel identical resources and propose a heuristic based on these results. [22] solve a PSP with harmonic periods, prohibiting tasks going over their periods, with zero jitter and latency constraints, and multi-rate precedence constraints defined on the job level. These constraints are represented by a function $code_{i,j}(k,l)$ that equals 1 if job $\tau_i^k$ should precede $\tau_j^l$. The authors propose an algorithm of latency marking to solve this problem that finds an optimal solution if one exists. Addressing this PSP on parallel identical resources and with no end-to-end latency constraint later, [23] prove the NP-hardness of the problem and propose a branch-and-bound approach and a heuristic algorithm. In the branch-and-bound, branch-and-cut, and branch-and-price algorithms, all the possible combinations of the task orderings are explored in a smart way to find the one that optimizes the given criterion.

For the PSP with zero jitter, [87] assume initial offsets called phases, where the first job of a task $\tau_i$ can be scheduled not earlier than at time $r_i$ and needs

Table 5: Works on non-preemptive periodic scheduling on a single resource

| Reference | Problem in the three-field notation | Results / Approaches |
|---|---|---|
| [88] | $1\|T_i, d_i \leq T_i\|-$ | chained windows heuristic |
| [58] | $1\|T_i, d_i = T_i+p_i-1, \overline{jit}_i = 0, \text{mltrt}\|-$ | sched. conditions |
| [76] | $1\|T_i, d_i = T_i, \overline{jit}_i = 0\|-$ | sched. condition, heuristic |
| [77] | $1\|T_i, d_i = T_i, \overline{jit}_i = 0\|-$ | sched. conditions |
| [126] | $1\|T_i, d_i = T_i, \overline{jit}_i = 0\|-$ | heuristic |
| [124] | $1\|T_i, d_i = T_i, \overline{jit}_i = 0, p_i = 1\|-$ | sched. condition (graph coloring) |
| [103] | $1\|T_i, d_i = T_i, p_i = 1, \text{mltrt}, \star\|-$ | SMT |
| [22] | $1\|T_i^{harm}, d_i \leq T_i, \overline{jit}_i = 0, \text{mltrt}, \overline{lat}_h\|-$ | algorithm of latency marking |

to be processed by $r_i+T_i$. In other words, the time windows of the jobs of the different tasks are desynchronised with each other. The authors propose a number of bin-packing-inspired heuristics.

[20] is one of a few works that consider a PSP with arbitrary jitter requirements. Additionally, the considered PSP involves cyclic multi-rate precedence relations and end-to-end latency constraints. Note that although the authors assume no explicit deadline constraints, the end-to-end latency constraint (10) bounds the start time implicitly. The problem is solved by a simulated annealing algorithm ([116]).

Addressing a basic PSP on a single resource with no jitter constraints, [88] propose a heuristic algorithm reducing the problem complexity for the price of optimality by working with groups of jobs called chained windows. These are ordered subsets of jobs executed after each other that allow a limited number of jobs scheduled between them. Chained windows contribute to removing a large number of possible job orderings from consideration and thereby pruning the search tree that can be very big.

Working with a PSP on parallel identical resources without jitter constraints and with arbitrary release times, [67] present a necessary and sufficient sched. condition of two tasks. The authors also provide a condition when two tasks without jitter requirements can be scheduled as one task with zero jitter requirements, and use it in the proposed heuristic. They solve the problem by first decomposing the task set to subsets that can be scheduled independently and then solve multiple independent sub-problems. [101] proposes a heuristic solution to solve the PSP without jitter requirements by first pairwise clustering the tasks and then assigning each cluster to a resource, and scheduling it afterwards.

[101] uses the graph expansion proposed earlier in [102], where a task $\tau_i$ in the directed acycic graph of the precedence relations, as in Figure 6a, is substituted by $n_i$ jobs. [104] address the same PSP proposing to tighten the time windows where tasks can be scheduled using their precedence relations. Further, they

Table 6: Works on non-preemptive periodic scheduling on parallel identical resources

| Reference | Problem in the three-field notation | Results / Approaches |
|---|---|---|
| [119] | $\mathrm{P}\|T_i, r_i, d_i, \mathrm{prec}, \mathrm{mlrt}, \star\|T_{max}$ | local search heuristic |
| [99] | $\mathrm{P}\|T_i, r_i, d_i \leq r_i{+}T_i, \mathrm{mltrt}, \star\|R$ | CP |
| [67] | $\mathrm{P}\|T_i, r_i, d_i = r_i{+}T_i\|R$ | sched. conditions, heuristic |
| [87] | $\mathrm{P}\|T_i, r_i, d_i = r_i{+}T_i, \overline{jit}_i = 0, \star\|-$ | heuristics (bin packing) |
| [12] | $\mathrm{P}\|T_i, r_i, d_i = T_i, \mathrm{prec}, \star\|-$ | ILP, heuristics |
| [20] | $\mathrm{P}\|T_i, r_i, \overline{jit}_i, \mathrm{mltrt}, \overline{lat}_h\|-$ | simulated annealing |
| [14] | $\mathrm{P}\|T_i, d_i \leq T_i, \mathrm{prec}, \star\|R$ | SMT, decomposition |
| [86] | $\mathrm{P}\|T_i, d_i = T_i, \mathrm{prec}\|T_{max}$ | genetic algorithm |
| [92] | $\mathrm{P}\|T_i, d_i = T_i, \mathrm{prec}\|-$ | heuristics (Worst-Fit) |
| [101] | $\mathrm{P}\|T_i, d_i = T_i, \mathrm{prec}, \star\|-$ | heuristic (pairwise clustering) |
| [104] | $\mathrm{P}\|T_i, d_i = T_i, \mathrm{prec}, \star\|-$ | heuristic |
| [33] | $\mathrm{P}\|T_i, d_i = T_i, \mathrm{prec}, \star\|-$ | genetic algorithm, heuristic |
| [18] | $\mathrm{P}\|T_i, d_i = T_i, \overline{jit}_i = 0\|R$ | sched. condition, heuristic |
| [56] | $\mathrm{P}\|T_i, d_i = T_i, \overline{jit}_i = 0\|R$ | sched. conditions, heuristic |
| [107] | $\mathrm{P}\|T_i, d_i = T_i, \overline{jit}_i = 0\|R$ | branch-and-price, heuristic |
| [23] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, \mathrm{mltrt}\|R$ | heuristic, branch-and-bound |
| [97] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, l_{i,j}\|\alpha$ | sched. condition,<br>ILP, heuristic |
| [55] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, \mathrm{mltrt}\|-$ | sched. condition, heuristic |
| [109] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0,$<br>$\mathrm{chains}, \overline{lat}_h, \star\|\sum lat_h$ | ILP |
| [68] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0\|R$<br>$\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, \mathrm{mlrt}\|R$ | sched. condition Eq. (21),<br>approx. algorithms |
| [66] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0\|R$ | heuristic, partitioning |
| [93] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0,$<br>$p_i = 1, \mathrm{mlrt}\|R$ | ILP, decomposition |
| [31] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0\|R$ | bin trees, approx. algorithms |
| [32] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, \star\|R$ | integer programming |
| [3] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, \star\|\alpha$ | ILP, heuristic (game theory) |
| [118] | $\mathrm{P}\|T_i, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0, p_i = 1\|-$ | sufficient sched. condition,<br>Periodic Maintenance Scheduling |
| [75] | $\mathrm{P}\|T_i, \mathrm{prec}, \overline{lat}_h, \star\|-$ | SMT, decomposition |
| [16] | $\mathrm{P}\|T_i^{harm}, d_i = T_i\|R$ | First-Fit heuristic |
| [28] | $\mathrm{P}\|T_i^{harm}, d_i = T_i{+}p_i{-}1, \overline{jit}_i = 0,$<br>$\overline{lat}_h, \mathrm{mltrt}\|-$ | branch-and-bound, heuristic |
| [29] | $\mathrm{P}\|T_i^{pow2}, r_i, d_i \leq T_i, \overline{jit}_i = 0, \star\|C_{max}$ | First-Fit based heuristic |
| [73] | $\mathrm{P}\|T_i^{pow2}, d_i = T_i, \overline{jit}_i = 0\|R$ | transformation to bin packing,<br>ILP using bin packing,<br>simulated annealing |
| [50] | $\mathrm{P}\|T_i^{pow2}, d_i = T_i, \overline{jit}_i = 0, \star\|-$ | SMT, heuristic |

use it in the proposed heuristic.

Multiple works solve problems in particular domains. [28] and [99] focus on avionic IMA architectures. [12] and [92] assume AUTOSAR automotive applications. On the other hand, [33] and [86] position themselves to solve PSPs with a general application domain.

Some authors deal with scheduling of networks. These works often not only have very specific platform-dependent constraints (such as memory requirements or size of a buffer of switches in TTEthernet networks), but they are also difficult to categorize in terms of the three-field notation introduced in Section 2. [73] and [29] address a PSP arising from the scheduling of a FlexRay network. In this network, the messages are sent in 64 cycles, each consisting of a predefined number of slots, and multiple messages can be scheduled in each slot in each cycle. Thus, this problem can be seen as scheduling on parallel identical resources (slots) with periods being powers of 2. [73] present a transformation of this PSP to a specific version of the 2D bin packing problem shown in Section 4, where the width of a bin is the maximal payload in bytes and its height is the number of cycles. Whereas [73] and [107] consider the minimization of the used number of slots (number of bins in the bin packing problem), [29] minimize the maximum makespan (completion time of the latest task) over all slots and all cycles. Additionally, [29] has release times and general deadlines that are considered to be on the cycle edges and require the same signal to be scheduled at the same place at multiple schedules for different FlexRay networks. On the other hand, [103] schedule messages in multi-hop wireless networks allowing retransmissions.

Finally, [109], [75], [50], and [14] solve the problem of finding a route for the messages in switched networks along with the scheduling problem. [109] and [75] consider a TTEthernet network, and [50] and [14] solve the problem for networks-on-chip. Whereas [109] address a PSP minimizing the sum of the end-to-end latencies and solve the joint scheduling and routing problem with ILP, [75] apply an SMT approach with a possible decomposition of the scheduling in separate time slices at the price of the solution optimality. [50] and [14] also apply SMT, the former using the transformation to 2D bin packing problem and the latter considering multiple optimization criteria that include equalizing the distribution of the resources utilization and the minimization of the number of used resources.

### 5.1.2 Scheduling on Dedicated Resources and on Heterogeneous Resources

Tables 7 and 8 present works in this subsection. Several articles deal with the scheduling of time-triggered networks. [110] is a pioneering work addressing the scheduling of multi-hop networks (e.g., TTEthernet). The author presents an SMT formulation for the PSP with harmonic periods, zero jitter, precedence, and end-to-end latency constraints, additionally introducing bounded switch memory constraints and simultaneous relay constraints. [91] describes the scheduling problem of time-sensitive networks under the time-triggered com-

munication paradigm IEEE 802.1Qbv with array theory encoding using SMT to solve the problem.

Combining the scheduling of networks with the scheduling of processing units helps to achieve a more flexible solution. [15] and [49] solve PSPs arising in avionic domains with the former proposing a scalable optimal integer programming solution and the latter proposing a heuristic addressing the PSP, in which some tasks are zero-jitter while others are not. In the automotive domain, [125] consider the scheduling of a distributed system comprised of processing units connected by a time-triggered Ethernet network with two objective functions: end-to-end latency and response time (i.e., the time when the last task in the transaction finishes its execution) minimization, and their weighted combination. Further, [83] consider a multicore on-chip architecture connected by a bus and explore the influence of jitter requirements on the solution feasibility. The authors reveal that zero-jitter constraints on tasks can result in up to 28% lower achievable resource utilization compared to reasonably constrained task jitter. [117] also works in the automotive domain. The authors deal with Time-Sensitive Networks (TSN) scheduling, proposing an improvement of TSN switches that results in higher task schedulability.

[108], [98], [113], [1], and [37] solve PSPs with no specific domain. [108] present a tutorial-like article presenting a CP approach to solve the PSP with multi-rate precedence constraints. [98] consider a combination of scheduling and voltage scaling problems on heterogeneous resources and propose a constraint logic programming approach. [113] assume heterogeneous TTEthernet-based mixed-criticality systems. Finally, [37] schedule processing units connected by a network-on-chip.

Control performance optimization is considered by [42], [105], and [84]. [42] and [105] combine the scheduling problem with the problem of setting the periods of control plants corresponding to transactions. The approaches proposed in these works allow one to set the periods to pre-defined values and to only solve the scheduling problem considered in this article. On the other hand, [84] optimize the control performance depending on the end-to-end latency with fixed task periods only.

[106] and [13] address the problem of scheduling a set of new tasks such that the change in the existing schedule is minimal. Whereas [106] prohibit changing the end-to-end latency of the scheduled transactions, [13] keep the schedule of "old" tasks untouched to minimize the integration cost.

Addressing PSPs on heterogeneous resources, [102] considers the minimization of the sum of the communication costs. The author proposes the above-mentioned graph expansion used in a two-stage heuristic, where the first stage defines the task mapping, whereas the second one deals with their scheduling. [95] look at the PSP with zero release times, zero jitter requirements, optimizing the transaction response times and solve the problem by the branch-and-bound approach. To the best of our knowledge, it is the only work to allow non-integer start times $s_i^k$ of tasks. Finally, [57] propose a heuristic without backtracking and an optimal branch-and-cut algorithm for the PSP with zero jitters and multi-rate precedence constraints.

Table 7: Works on non-preemptive PSPs on dedicated resources

| Reference | Problem in the three-field notation | Results / Approaches |
|---|---|---|
| [15] | $\text{PD}\|T_i, r_i, d_i, \overline{jit}_i = 0, l_{i,j}, \star\|-$ | integer programming |
| [117] | $\text{PD}\|T_i, r_i, d_i \leq T_i, \overline{jit}_i = 0, \text{prec}, \star\|lat_{max}$ | ILP |
| [1] | $\text{PD}\|T_i, d_i, \text{prec}, \star\|T_{max}$ | branch-and-bound |
| [49] | $\text{PD}\|T_i, d_i, \overline{jit}_i = 0, \text{prec}, \star\|-$ | heuristics |
| [91] | $\text{PD}\|T_i, d_i = T_i, \overline{jit}_i, \text{prec}, \overline{lat}_h, \star\|\sum w_i \cdot jit_i$ | SMT |
| [13] | $\text{PD}\|T_i, d_i = T_i, \overline{jit}_i = 0, l_{i,j}, \overline{lat}_h, \star\|$integr. cost | SMT |
| [105] | $\text{PD}\|T_i, d_i = T_i, \overline{jit}_i = 0,$ chains, $\overline{lat}_h, \star\|$ctrl | ILP, decompostion |
| [83] | $\text{PD}\|T_i, d_i = 2 \cdot T_i, \overline{jit}_i, \text{prec}\|-$ | ILP, SMT, heuristic |
| [108] | $\text{PD}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0,$ mltrt, $\overline{lat}_h, \star\|-$ | CP |
| [42] | $\text{PD}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0,$ chains, $\overline{lat}_h, \star\|$ctrl | ILP |
| [125] | $\text{PD}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0,$ chains, $\overline{lat}_h, \star\|lat_{max}$ | MIP |
| [106] | $\text{PD}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0,$ prec, $\overline{lat}_h, \|$integr. cost | SMT, decomposition |
| [84] | $\text{PD}\|T_i, \overline{jit}_i = 0, \text{prec}, \overline{lat}_h\|ctrl$ | CP, ILP, heuristic |
| [37] | $\text{PD}\|T_i^{harm}, r_i, d_i = r_i + T_i, \overline{jit}_i = 0, \text{mltrt}, \star\|-$ | heuristic, SMT |
| [110] | $\text{PD}\|T_i^{harm}, d_i = T_i + p_i - 1, \overline{jit}_i = 0, l_{i,j}, \overline{lat}_h, \star\|-$ | SMT, decomposition |

Table 8: Works on non-preemptive PSPs on heterogeneous resources

| Reference | Problem in the three-field notation | Results / Approaches |
|---|---|---|
| [113] | $\text{RD}\|T_i, d_i, \text{prec}, \star\|-$ | tabu search |
| [98] | $\text{RD}\|T_i, d_i \leq T_i, \text{prec}, \star\|$energy | constraint logic programming |
| [102] | $\text{R}\|T_i, d_i = T_i, \text{mltrt}, \star\|$comm. cost | heuristic (graph expansion) |
| [95] | $\text{R}\|T_i, d_i = T_i, \text{mltrt}\|$resp. | branch-and-bound, heuristic |
| [57] | $\text{R}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0, \text{mltrt}\|-$ | greedy heuristic, branch-and-cut |

## 5.2 Works on a Preemptive PSP

Table 9 presents works in this subsection. All the presented articles solving preemptive PSPs consider general periods, and most of the works assume precedence constraints. [122] look at a PSP on a single resource and minimize the number of preemptions by setting the cost of one time unit for each preemption. The authors put zero jitter requirements defining them for the start times of the first chunks of the task's jobs and propose schedulability conditions based on the task periods and processing times. This condition is used in a proposed heuristic approach, in which tasks are scheduled in the non-decreasing order of periods.

For a PSP on parallel identical resources, [70] propose an ant colony optimization approach addressing the PSP under the migration policy, where the preempted job may change the resource. [114] apply a Timed Petri Net model [79] to solve the PSP with precedence constraints, but without migration.

Considering a dedicated resources environment, [36] look at a PSP with arbitrary release times, deadlines, and precedence relations. The authors focus on the case of deadlines larger than the periods. Unlike other approaches, the schedule length is a decision variable (specifically, it is a multiple of the least common multiple of the periods) since it results in a higher number of feasible problem instances due to the broader search space. The authors propose a constructive modified list scheduling algorithm with limited backtracking to solve this problem. Similarly to the previous work, [90] proposes a genetic algorithm to solve a PSP with arbitrary release times and precedence constraints. However, this work considers the time lags, end-to-end latency constraints, and no explicitly stated deadlines.

[96] solve a PSP on dedicated resources with the minimization of the maximum normalized latency over all transactions, i.e., their end-to-end latency divided by the length of the time window that they may be scheduled in. The authors assume work-conserving scheduling - if there are jobs ready to be scheduled at any time, at least one must be scheduled. [74] propose an ILP model for both non-preemptive and preemptive scheduling on the processing units and FlexRay network, allowing a job to preempt another job only once. The authors consider the minimization of the sum of the end-to-end latencies.

If the number of preemption and preemption points are not limited, preemptive PSPs $\cdots|T_i, \cdots, \text{ pmtn}|\cdots$ can be solved by the same approach as the non-preemptive PSPs with each task $\tau_i$ split into $p_i$ sub-tasks of a unit time, i.e., $\cdots|T_i, p_i = 1, \cdots|\cdots$. However, it drastically increases the number of variables and constraints in the model. [21], uses this trick and proposes an SMT and ILP model and an SMT-based heuristic.

Finally, the following works deal with heterogeneous resources. A modified list scheduling heuristic is proposed in [17], solving the PSP with arbitrary release times, deadlines, and precedence and end-to-end latency constraints. In this work, both the preemptive and non-preemptive tasks are handled, and the number of preemptions is minimized. Addressing the same problem without

Table 9: Works on preemptive PSPs

| Reference | Problem in the three-field notation | Results / Approaches |
|---|---|---|
| [122] | $1\|T_i, d_i = T_i, \overline{jit}_i = 0, \text{mltrt}, \text{pmtn}\|n_{pmtn}$ | sched. conditions, heuristic |
| [78] | $\text{P}\|T_i, r_i, d_i \leq r_i + T_i, \text{mlrt}, \text{pmtn}\|-$ | simulated annealing |
| [70] | $\text{P}\|T_i, r_i, d_i, \text{mlrt}, \text{pmtn}\|n_{pmtn}$ | ant colony optimization |
| [114] | $\text{P}\|T_i, r_i, d_i, \text{prec}, \star, \text{pmtn}\|\text{energy}$ | Petri Nets |
| [36] | $\text{PD}\|T_i, r_i, d_i, \text{prec}, \star, \text{pmtn}\|C_{max}$ | heuristic (limit backtrack) |
| [96] | $\text{PD}\|T_i, r_i, \text{mltrt}, \text{pmtn}\|\text{resp.}$ | heuristics |
| [90] | $\text{PD}\|T_i, l_{i,j}, \star, \text{pmtn}\|T_{max}$ | Genetic Algorithm |
| [74] | $\text{PD}\|T_i, d_i = T_i + p_i - 1, \overline{jit}_i = 0, \text{prec}, \overline{lat}_h, \star, \text{pmtn}\|\sum lat_h$ | ILP |
| [21] | $\text{PD}\|T_i, d_i = T_i, \overline{jit}_i = 0, \text{prec}, \overline{lat}_h, \star, \text{pmtn}\|\sum lat_h$ | SMT, heuristic, MIP |
| [17] | $\text{R}\|T_i, r_i, d_i, \text{prec}, \overline{lat}_h, \star, \text{pmtn}\|n_{pmtn}$ | heuristic (list scheduling) |
| [112] | $\text{R}\|T_i, r_i, d_i, \text{prec}, \star, \text{pmtn}\|-$ | branch-and-bound |
| [71] | $\text{R}\|T_i, r_i, d_i = r_i + T_i, \text{prec}, \star, \text{pmtn}\|n_{pmtn}$ | CP |

end-to-end latency constraints, [112] propose a tuned branch-and-bound algorithm with the pruning of infeasible search-tree paths and symmetry avoidance. The solution allows the system to be in two modes: either preemptive or non-preemptive. A PSP as a Constraint Satisfaction Problem is further formulated by [71], where both preemptive and non-preemptive tasks can be handled, and the number of preemptions is minimized.

## 5.3 Related Problems

This subsection discusses problems that are similar to the PSP, since some results for these problems can be useful for (specific) PSPs. There are three classes of problems in this section: 1) problems more general in some parameters, for which the PSP is a particular case (distance-constrained scheduling); 2) problems with the same optimal solution as a PSP in specific cases (pinwheel scheduling or windows scheduling). This class is also comprised of problems that allow changing period values with the aim of setting the period values as close as possible to the the ideal ones (perfectly periodic scheduling [94], [9], and [8]); and 3) problems equivalent to specific PSPs that are called differently (periodic maintenance scheduling, chairmen assignment, scheduling with release times and deadlines).

In *the distance-constrained scheduling problem* addressed in [80], a job of each task $\tau_i$ should be executed not earlier than $l_i$ time units and no later than $u_i$ time units from the execution of the previous job and has a unit processing time. Thus, unlike the PSP, where the scheduling windows are fixed and independent

of the task start times, in this problem, the scheduling window for each job is related to the start time of the previous job. However, for $l_i = u_i = T_i$, the problem is equivalent to $1|T_i, \overline{jit}_i = 0, p_i = 1|-$.

*The periodic maintenance scheduling problem* considered in [118] is the case with $l_i = u_i = T_i$ by the problem definition. Thus, as mentioned in the previous subsection, it is equivalent to the PSP with zero jitter requirements and unit processing times, $P|T_i, \overline{jit}_i = 0, p_i = 1|-$ and belongs to the third class. The authors state the sufficient condition of a task set schedulability for $m$ parallel identical resources formulated in Inequality (30).

$$\sum_{i=1}^{j-1} \frac{1}{\gcd(T_i, T_j)} < m, \qquad j = 2, \cdots, n. \tag{30}$$

*The Pinwheel Scheduling Problem* [48] (also called Windows Scheduling Problem) belongs to the second class of problems, while defining only the $u_i$ parameter. Accordingly, each job of each task $\tau_i$ has to be scheduled no later than $u_i$ time units from the execution of the previous job, also having a unit processing time. However, unlike the periodic maintenance scheduling problem, there cannot be empty slots in the solution, i.e., at each time moment some task must be scheduled. When minimizing the number of jobs, the solution of the pinwheel problem can be the solution of a PSP with zero jitter requirements $1|T_i, \overline{jit}_i = 0, p_i = 1|-$, if a feasible solution to the corresponding PSP exists and it does not contain empty time slots.

Furthermore, in *the chairmen assignment problem* [115], one requires the fairness of the schedule: the number of scheduled task jobs before each time moment should be proportional to its frequency equal to the inverse of the period. Formally, at every time $t$ task $\tau_i$ must have been scheduled either $\left\lfloor \frac{1}{T_i} \cdot t \right\rfloor$ or $\left\lceil \frac{1}{T_i} \cdot t \right\rceil$ times. This problem is equivalent to a PSP with deadline equal to periods and unit processing times, i.e., $P|T_i, d_i = T_i, p_i = 1|-$, since the aforementioned condition requires that each task should be scheduled once per its period.

Finally, a PSP can be solved by methods applied to solve scheduling problems with arbitrary release times and deadlines $\cdots|r_i, d_i, \cdots|\cdots$ [120]. However, the jitter requirements are not considered here similarly to the chairmen assignment problem. Note that the scalability of a PSP can be higher due to the decreased number of entities one is working with compared to the non-periodic problem with arbitrary release times and deadlines.

# 6 Conclusions

In this paper, we present the basic principles and related works to time-triggered periodic scheduling of hard real-time systems. Many real-life problems have a periodic nature, and therefore, their applications can be found in versatile fields, such as embedded systems, maintenance, and production. These scheduling

problems are often presented without systematic abstraction, and many authors merge the problem statement and its solution. We address this issue by extending the three-field Graham notation to define periodic scheduling problems (PSPs). To facilitate an understanding and motivate future research, we provide straightforward implementations of three PSPs in the Satisfiability Modulo Theories formalism with the source code. The development of an efficient scheduling algorithm is hard, as one can see from the complexity results listing the few polynomially solvable PSPs. To help this development, we provide an overview of the simple state-of-the-art methods and tricks to solve PSPs. Finally, we survey works on periodic scheduling, categorizing them with an extended three-field notation.

## Acknowledgments

This work was supported by the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation program under grant agreement No. 832011 (THERMAC) and
by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).

## References

[1] Tarek F Abdelzaher and Kang G Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on parallel and distributed systems*, 10(11):1179–1191, 1999.

[2] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. An empirical survey-based study into industry practice in real-time systems. In *Real-Time Systems Symposium (RTSS)*. IEEE, 2020.

[3] Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik, and Balakrishna J Prabhu. Strictly periodic scheduling in IMA-based architectures. *Real-Time Systems*, 48(4):359–386, 2012.

[4] Amos Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded world*, 2004:235–252, 2004.

[5] Muhammad Ali Awan, Pedro F Souto, Benny Akesson, Konstantinos Bletsas, and Eduardo Tovar. Uneven memory regulation for scheduling IMA applications on multi-core platforms. *Real-Time Systems*, 55(2):248–292, 2019.

[6] Egon Balas. Project scheduling with resource constraints. Technical report, Carnegie-Mellon Univ Pittsburgh Pa, 1968.

[7] Amotz Bar-Noy, Randeep Bhatia, Joseph (Seffi) Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research*, 27(3):518–544, 2002.

[8] Amotz Bar-Noy, Vladimir Dreizin, and Boaz Patt-Shamir. Efficient algorithms for periodic scheduling. *Computer Networks*, 45(2):155–173, 2004.

[9] Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. Nearly optimal perfectly periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.

[10] Jeffrey R Barker and Graham B McMahon. Scheduling the general job-shop. *Management Science*, 31(5):594–598, 1985.

[11] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems*, 2(4):301–324, 1990.

[12] Matthias Becker, Dakshina Dasari, Borislav Nicolic, Benny Akesson, Vincent Nélis, and Thomas Nolte. Contention-free execution of automotive applications on a clustered many-core platform. *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24, 2016.

[13] Sofiene Beji, Sardaouna Hamadou, Abdelouahed Gherbi, and John Mullins. Smt-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures. *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pages 165–174, 2014.

[14] Alexander Biewer, Benjamin Andres, Jens Gladigau, Torsten Schaub, and Christian Haubelt. A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 357–362, San Jose, CA, USA, 2015. EDA Consortium.

[15] Mathias Blikstad, Emil Karlsson, Tomas Lööw, and Elina Rönnberg. An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system, 2017.

[16] Yang Cai and MC Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 15(6):572–599, 1996.

[17] Thomas Carle, Dumitru Potop-Butucaru, Yves Sorel, and David Lesens. From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. *Leibniz Transactions on Embedded Systems*, 2(2):01–1, 2015.

[18] Jinchao Chen, Chenglie Du, Fei Xie, and Zhenkun Yang. Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems. *Real-Time Systems*, 52(3):239–271, 2016.

[19] Sheng Cheng, John A Stankovic, and Krithivasan Ramamritham. Scheduling algorithms for hard real-time systems–a brief survey. *Tutorial Hard Real-Time Systems*, pages 150–173, 1987.

[20] Sheng-Tzong Cheng and Ashok K Agrawala. Allocation and scheduling of real-time periodic tasks with relative timing constraints. *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, pages 210–217, 1995.

[21] Silviu S Craciunas and Ramon Serna Oliver. Smt-based task-and network-level static schedule generation for time-triggered networked systems. In *Proceedings of the 22nd international conference on real-time networks and systems*, page 45. ACM, 2014.

[22] Liliana Cucu, Remy Kocik, and Yves Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. *Proceedings of 10th Real-Time Systems Conference, RTS'02*, 2002.

[23] Liliana Cucu and Yves Sorel. Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints. *Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG*, 4, 2004.

[24] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, New Jersey, 1998.

[25] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):1–44, 2011.

[26] Federico Della Croce, Fabio Salassa, and Vincent T'kindt. A hybrid heuristic approach for single machine scheduling with release times. *Computers & Operations Research*, 45:7–11, 2014.

[27] Jitender S Deogun and MC Kong. On periodic scheduling of time-critical tasks. *IFIP Congress*, pages 791–796, 1986.

[28] Emilie Deroche, Jean-Luc Scharbarg, and Christian Fraboul. A greedy heuristic for distributing hard real-time applications on an IMA architecture. In *Proceedings of the 2017 12th IEEE International Symposium on Industrial Embedded Systems*, pages 1–8, Toulouse, France, 2017. IEEE.

[29] Jan Dvorak and Zdenek Hanzalek. Multi-variant time constrained FlexRay static segment scheduling. *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, pages 1–8, 2014.

[30] Martin E Dyer and Laurence A Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2-3):255–270, 1990.

[31] F. Eisenbrand, N. Hahnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese. Scheduling periodic tasks in a hard real-time environment. *Automata, Languages and Programming*, 6198:299–311, 2010.

[32] Friedrich Eisenbrand, Karthikeyan Kesavan, Raju S Mattikalli, Martin Niemeier, Arnold W Nordsieck, Martin Skutella, José Verschae, and Andreas Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. *European Symposium on Algorithms*, pages 11–22, 2010.

[33] Sebastien Faucou, A-M Deplanche, and J-P Beauvais. Heuristic techniques for allocating and scheduling communicating periodic tasks in distributed real-time systems. *Proceedings of 2000 IEEE International Workshop on Factory Communication System*, pages 257–265, 2000.

[34] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, 2008.

[35] Gerhard Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 152–161, Pisa, Italy, 1995. IEEE.

[36] Gerhard Fohler and Krithi Ramamritham. Static scheduling of pipelined periodic tasks in distributed real-time systems. In *Proceedings Ninth Euromicro Workshop on Real Time Systems*, pages 128–135. IEEE, 1997.

[37] Matthias Freier and Jian-Jia Chen. Time-triggered communication scheduling analysis for real-time multicore systems. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, Washington, DC, USA, 2015. IEEE.

[38] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*, volume 174. Freeman San Francisco, 1979.

[39] Celia A Glass. Feasibility of scheduling lot sizes of two frequencies on one machine. *European Journal of Operational Research*, 75(2):354–364, 1994.

[40] MJ Gonzalez and Jin W Soh. Periodic job scheduling in a distributed processor system. *IEEE Transactions on Aerospace and Electronic Systems*, AES-12(5):530–536, 1976.

[41] Raul Gorcitz, Emilien Kofman, Thomas Carle, Dumitru Potop-Butucaru, and Robert De Simone. On the scalability of constraint solving for static/off-line real-time scheduling. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 108–123. Springer, 2015.

[42] Dip Goswami, Martin Lukasiewycz, Reinhard Schneider, and Samarjit Chakraborty. Time-triggered implementations of mixed-criticality automotive software. *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1227–1232, 2012.

[43] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[44] Arpan Gujarati, Felipe Cerqueira, Björn B Brandenburg, and Geoffrey Nelissen. Correspondence article: a correction of the reduction-based schedulability analysis for apa scheduling. *Real-Time Systems*, 55(1):136–143, 2019.

[45] Claire Hanen and Zdenek Hanzalek. Periodic scheduling and packing problems. In *arXiv*, 2020.

[46] Zdenek Hanzalek and Claire Hanen. The impact of core precedences in a cyclic RCPSP with precedence delays. *J. Scheduling*, 18(3):275–284, 2015.

[47] Richard Hladík, Anna Minaeva, and Zdeněk Hanzálek. On the complexity of a periodic scheduling problem with precedence relations. In *The 14th Annual International Conference on Combinatorial Optimization and Applications (COCOA'20)*, 2020.

[48] Chih-Wen Hsueh and Kwei-Jay Lin. An optimal pinwheel scheduler using the single-number reduction technique. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 196–205, Washington, DC, 1996. IEEE.

[49] Menglan Hu, Jun Luo, Yang Wang, and Bharadwaj Veeravalli. Scheduling periodic task graphs for safety-critical time-triggered avionic systems. *Aerospace and Electronic Systems, IEEE Transactions on*, 51(3):2294–2304, 2015.

[50] Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. Static scheduling of a time-triggered network-on-chip based on SMT solving. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 509–514. IEEE, 2012.

[51] Damir Isovic and Gerhard Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 207–216, Orlando, FL, 2000. IEEE.

[52] Damir Isovic and Gerhard Fohler. Handling mixed sets of tasks in combined offline and online scheduled real-time systems. *Real-time systems*, 43(3):296–325, 2009.

[53] Tobias Jacobs and Salvatore Longo. A new perspective on the windows scheduling problem. *arXiv preprint arXiv:1410.7237*, 2014.

[54] Kevin Jeffay, Donald F Stanat, and Charles U Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *IEEE real-time systems symposium*, pages 129–139, US, 1991. IEEE.

[55] Omar Kermia. Optimizing distributed real-time embedded system handling dependence and several strict periodicity constraints. *Advances in Operations Research*, 2011, 2011.

[56] Omar Kermia. An efficient approach for the multiprocessor non-preemptive strictly periodic task scheduling problem. *Journal of Systems Architecture*, 79:31–44, 2017.

[57] Omar Kermia and Yves Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *Proceedings of ISCA 20th international conference on parallel and distributed computing systems (PDCS)*, Las Vegas, Nevada, USA, 2007.

[58] Omar Kermia and Yves Sorel. Schedulability analysis for non-preemptive tasks under strict periodicity constraints. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 25–32, Kaohsiung, Taiwan, 2008. IEEE.

[59] Jad Khatib, Alix Munier-Kordon, Enagnon Cédric Klikpo, and Kods Trabelsi-Colibet. Computing latency of a real-time system modeled by synchronous dataflow graph. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 87–96. ACM, 2016.

[60] Eun-Seok Kim and Celia A Glass. Perfect periodic scheduling for three basic cycles. *Journal of Scheduling*, 17(1):47–65, 2014.

[61] Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[62] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[63] Anis Kooli and Mehdi Serairi. A mixed integer programming approach for the single machine problem with unequal release dates. *Computers & Operations Research*, 51:323–330, 2014.

[64] Hermann Kopetz. Time-triggered real-time computing. *Annual Reviews in Control*, 27(1):3–13, 2003.

[65] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[66] Jan Korst, Emile Aarts, and Jan Karel Lenstra. Scheduling periodic tasks. *INFORMS journal on Computing*, 8(4):428–435, 1996.

[67] Jan Korst, Emile Aarts, and Jan Karel Lenstra. Scheduling periodic tasks with slack. *INFORMS Journal on Computing*, 9(4):351–362, 1997.

[68] Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic multiprocessor scheduling. In *Parallel Architectures and Languages Europe*, pages 166–178, Berlin, 1991. Springer.

[69] Yacine Laalaoui and Nizar Bouguila. Pre-run-time scheduling in real-time systems: Current researches and artificial intelligence perspectives. *Expert Systems with Applications*, 41(5):2196–2210, 2014.

[70] Yacine Laalaoui and Habiba Drias. Aco approach with learning for preemptive scheduling of real-time tasks. *International Journal of Bio-Inspired Computation*, 2(6):383–394, 2010.

[71] Erjola Lalo, Raphael Weber, Andreas Sailer, Juergen Mottok, and Christian Siemers. On solving task allocation and schedule generation for time-triggered LET systems using constraint programming. In *ARCS Workshop 2019; 32nd International Conference on Architecture of Computing Systems*, pages 1–8, Copenhagen,Denmark, 2019. VDE.

[72] Jean B Lasserre and Maurice Queyranne. Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. pages 136–149, USA, Pittsburgh, 1992.

[73] Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. FlexRay schedule optimization of the static segment. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 363–372, France, Grenoble, 2009. IEEE/ACM.

[74] Martin Lukasiewycz, Reinhard Schneider, Dip Goswami, and Samarjit Chakraborty. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *17th Asia and South Pacific Design Automation Conference*, pages 665–670, Australia, Sydney, 2012. IEEE.

[75] Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. Stability-aware integrated routing and scheduling for control applications in ethernet networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 682–687. IEEE, 2018.

[76] Mohamed Marouf and Yves Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period. In *18th International Conference on Real-Time and Network Systems RTNS'10*, pages 50—58, France, Toulouse, 2010.

[77] Mohamed Marouf and Yves Sorel. Scheduling non-preemptive hard real-time tasks with strict periods. In *Emerging Technologies and Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–8, Toulouse, France, 2011. IEEE.

[78] Shane D McLean, Silviu S Craciunas, Emil Alexander Juul Hansen, and Paul Pop. Mapping and scheduling automotive applications on adas platforms using metaheuristics. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 329–336. IEEE, 2020.

[79] Philip Merlin and David Farber. Recoverability of communication protocols-implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043, 1976.

[80] Philippe Michelon, Dominique Quadri, and Marcos Negreiros. On a class of periodic scheduling problems: Models, lower bounds and heuristics. In *2008 International Multiconference on Computer Science and Information Technology*, pages 899–906, Poland, Wisla, 2008. IEEE.

[81] Anna Minaeva. *Scalable Scheduling Algorithms for Embedded Systems with Real-Time Requirements*. PhD thesis, Czech Technical University in Prague, 2019.

[82] Anna Minaeva. Smt implementation of basic periodic scheduling problems using z3 solver, 2020.

[83] Anna Minaeva, Benny Akesson, Zdeněk Hanzálek, and Dakshina Dasari. Time-triggered co-scheduling of computation and communication with jitter requirements. *IEEE Transactions on Computers*, 67(1):115–129, 2017.

[84] Anna Minaeva, Debayan Roy, Benny Akesson, Zdenek Hanzalek, and Samarjit Chakraborty. Control performance optimization for application integration on automotive architectures. *IEEE Transactions on Computers*, 2020.

[85] Anna Minaeva, Přemysl Šůcha, Benny Akesson, and Zdeněk Hanzálek. Scalable and efficient configuration of time-division multiplexed resources. *Journal of Systems and Software*, 113:44 – 58, 2016.

[86] Yannick Monnier, Jean-Pierre Beauvais, and Anne-Marie Déplanche. A genetic algorithm for scheduling tasks in a real-time distributed system. In *Euromicro Conference, 1998. Proceedings. 24th*, pages 708–714, Sweden, Vasteras, 1998. IEEE.

[87] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multisource software on multicore automotive ECUs—combining runnable sequencing with task scheduling. *Industrial Electronics, IEEE Transactions on*, 59(10):3934–3942, 2012.

[88] Mitra Nasri and Björn B Brandenburg. Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–86, USA, Pittsburgh, 2017. IEEE.

[89] Jerzy R Nawrocki, Adam Czajka, and Wojciech Complak. Scheduling cyclic tasks with binary periods. *Information processing letters*, 65(4):173–178, 1998.

[90] Roman Nossal. An evolutionary approach to multiprocessor scheduling of dependent tasks. *Future Generation Computer Systems*, 14(5-6):383–392, 1998.

[91] Ramon Serna Oliver, Silviu S Craciunas, and Wilfried Steiner. IEEE 802.1 Qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24. IEEE, 2018.

[92] Miloš Panić, Sebastian Kehr, Eduardo Quiñones, Bert Boddecker, Jaume Abella, and Francisco J Cazorla. Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–10, USA, New York, 2014. ACM.

[93] Kyung S Park and Doek K Yun. Optimal scheduling of periodic activities. *Operations Research*, 33(3):690–695, 1985.

[94] Shailesh Patil and Vijay K Garg. Adaptive general perfectly periodic scheduling. *Information processing letters*, 98(3):107–114, 2006.

[95] D-T Peng and Kang G Shin. Static allocation of periodic tasks with precedence constraints in distributed real-time systems. In *[1989] Proceedings. The 9th International Conference on Distributed Computing Systems*, pages 190–198, USA, CA, Newport Beach, 1989. IEEE.

[96] D-T Peng and Kang G Shin. Optimal scheduling of cooperative tasks in a distributed system using an enumerative method. *IEEE Transactions on Software Engineering*, 19(3):253–267, 1993.

[97] Clément Pira and Christian Artigues. Line search method for solving a non-preemptive strictly periodic scheduling problem. *Journal of Scheduling*, 19(3):227–243, 2016.

[98] Paul Pop, Kåre Harbo Poulsen, Viacheslav Izosimov, and Petru Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 233–238, 2007.

[99] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

[100] Diogo Quintas and Vasilis Friderikos. Energy efficient spatial tdma scheduling in wireless networks. *Computers & Operations Research*, 39(9):2091–2099, 2012.

[101] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *Parallel and Distributed Systems, IEEE Transactions on*, 6(4):412–420, April 1995.

[102] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings., 10th International Conference on Distributed Computing Systems*, pages 108–115, France, Paris, 1990. IEEE.

[103] Jin Woo Ro, Partha Roop, and Avinash Malik. Schedule synthesis for time-triggered multi-hop wireless networks with retransmissions. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 94–101, New Zealand, Auckland, 2015. IEEE.

[104] Stefan Ronngren and Behrooz A Shirazi. Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs. In *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences*, volume 2, pages 143–152, USA, HI, Wailea, 1995. IEEE.

[105] Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, and Samarjit Chakraborty. Multi-objective co-optimization of FlexRay-based distributed control systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, Austria, Vienna, 2016. IEEE.

[106] Florian Sagstetter, Peter Waszecki, Sebastian Steinhorst, Martin Lukasiewycz, and Samarjit Chakraborty. Multischedule synthesis for variant management in automotive time-triggered systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):637–650, 2015.

[107] Thijs Schenkelaars, Bart Vermeulen, and Kees Goossens. Optimal scheduling of switched FlexRay networks. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6, 2011.

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

[109] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. ILP-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 8–17, 2017.

[110] W. Steiner. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, 31:375–384, November 2010.

[111] Premysl Sucha and Zdenek Hanzalek. Deadline constrained cyclic scheduling on pipelined dedicated processors considering multiprocessor tasks and changeover times. *Mathematical and Computer Modelling*, 47(9):925 – 942, 2008.

[112] Ali Syed and Gerhard Fohler. Efficient offline scheduling of task-sets with complex constraints on large distributed time-triggered systems. *Real-Time Systems*, 55(2):209–247, 2019.

[113] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 473–482, Finland,Tampere, 2012. ACM.

[114] Eduardo Tavares, Raimundo Barreto, Meuse Oliveira Junior, Paulo Maciel, Marilia Neves, and Ricardo Lima. An approach for pre-runtime scheduling in embedded hard real-time systems with power constraints. In *16th Symposium on Computer Architecture and High Performance Computing*, pages 188–195, Brazil, Foz do Iguacu, 2004. IEEE.

[115] Robert Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32(3):323–330, 1980.

[116] Peter JM Van Laarhoven and Emile HL Aarts. *Simulated annealing: theory and applications*. Springer, 1987.

[117] Marek Vlk, Zdeněk Hanzálek, Kateřina Brejchová, Siyu Tang, Sushmit Bhattacharjee, and Songwei Fu. Enhancing schedulability and throughput of time-triggered traffic in ieee 802.1 qbv time-sensitive networks. *IEEE Transactions on Communications*, 2020.

[118] W.D. Wei and C.L. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2(2):90–93, 1983.

[119] Jia Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on software engineering*, 19(2):139–154, 1993.

[120] Jia Xu and David Lorge Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on software engineering*, 16(3):360–369, 1990.

[121] Jia Xu and David Lorge Parnas. On satisfying timing constraints in hard-real-time systems. *ACM SIGSOFT Software Engineering Notes*, 16(5):132–146, 1991.

[122] Patrick Meumeu Yomsi and Yves Sorel. *Schedulability analysis for non necessarily harmonic real-time systems with precedence and strict periodicity constraints using the exact number of preemptions and no idle time.* PhD thesis, INRIA, 2008.

[123] Sophia A Zelenova and Sergey V Zelenov. Schedulability analysis for strictly periodic tasks in rtos. *Programming and Computer Software*, 44(3):159–169, 2018.

[124] Sophia Anatolievna Zelenova and Sergey Vadimovich Zelenov. Non-conflict scheduling criterion for strict periodic tasks. *Proceedings of the Institute for System Programming of the RAS*, 29(6):183–202 (In Russian, 2017.

[125] Licong Zhang, Debkalpa Goswami, Reinhard Schneider, and Shiladri Chakraborty. Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 119–124, Singapore, SunTec, 2014. IEEE.

[126] Tianyu Zhang, Nan Guan, Qingxu Deng, and Wang Yi. Start time configuration for strictly periodic real-time task systems. *Journal of Systems Architecture*, 66:61–68, 2016.