

# Non-preemptive Scheduling of Distance Constrained Tasks Subject to Minimizing Processor Load

Klaus H. Ecker

Ohio University, Athens, OH, USA, ecker@ohio.edu

Alexander Hasenfuss

Clausthal University of Technology, Clausthal, Germany, hasenfuss@in.tu-clausthal.de

---

Scheduling problems that involve distance constrained tasks with repetitive requests are enjoying an increase in popularity in real time system design. Timing constraints defined by periods of fixed lengths for each task enforce some frequency of task execution. Alternatively deadlines could be specified relatively to the finish time of the previous execution of the same task. Hence, the scheduling algorithms deal with tasks that virtually consist of infinite chains of jobs with specified temporal distance constraints between each pair of adjacent jobs. In this paper, unlike approaches that consider a preemptive distance constrained task scheduling subject to minimize time distance between task execution, we consider a problem of non preemptive tasks systems with objective to reduce processor utilization while keeping the response times within their limits. As a consequence, free processor capacity remains to process aperiodic tasks. Moreover, as a side effect, the way the tasks get scheduled to obtain minimal processor utilization results in jitter reduction improving system stability.

*Keywords:* distance constrained tasks, relative timing constraints, minimizing processor utilization

---

## 1. Introduction

In real-time systems for controlling some environment such as facilities, power plants, and as well in embedded systems for mechatronics applications, there are generally many processes that must be executed repeatedly. The specification for such processes contains several conditions including the pace at which a process has to be performed periodically, or, similarly, the requirement of a repeated processing guided by a given maximal time lag. One of the first results for scheduling preemptive periodic tasks was the fundamental paper of Liu and Layland [23]. A periodic task in this context is a computation that has to be performed repeatedly; the time axis is partitioned in equal length intervals, and in each interval one task instance is processed. Since then a lot of work has been invested to study variations of the original problem, cf. [1].

The periodic task model is the common choice in today's commercial real time systems; mostly based on fixed priority schemes they are suited for many applications. However, there are situations where absolute periods do not fully capture the nature of the environment; at that point relative schemes come into play.

There are many approaches documented, Chen et al. [3] assume periodic tasks together with the additional, as they call it, relative timing constraints, a *low and high jitter* for the distance between two consecutive jobs. Other generalizations regard processing times. Mok et al. [24] consider a multiframe model where the tasks are instantiated periodically, but with different execution times in each interval. The model discussed in [9, 5] assumes the periodic execution of a set of tasks with precedences and relative timing constraints in form of min/max conditions between start and finish times of any two instances. Furthermore, upper and lower bounds for job execution times are assumed. A similar model is *end-to-end scheduling*, as considered e.g. by Gerber et al. [13, 14, 15], which deals with the scheduling of sets of tasks with precedences, deadlines, various constraints, and communication delays. Another model modification is discussed in [6] where repeating processes are considered, and the time between the processes is determined at each iteration step by a so-called dynamic temporal controller. In [4], Choi and Agrawala distinguish between static periodic processes and processes arriving dynamically (called aperiodic or sporadic). *Aperiodic* tasks are characterized by a given arrival time, a release (or ready) time, a deadline, and a worst case

execution time. *Multiple instance tasks* which are divided into periodic tasks and sporadic tasks are repeatedly invoked. The behavior of the classical periodic model in presence of aperiodic tasks is analyzed in [2]. Aperiodic tasks without deadlines in dynamic priority systems are considered in [26]. A *sporadic task* is characterized as a sequence of task instances (also called jobs). Between two consecutive jobs executions there is a given minimum inter-arrival time  $\delta$ , meaning that if an instance is invoked at some time  $t$ , the next instance is invoked at any time  $t'$  satisfying  $t' \geq t + \delta$ . This way it is ensured that the frequency of sporadic occurrences is bounded above. It is assumed that the sporadic tasks are included dynamically, i.e. on-line at run-time, into the schedule.

Many technical systems have distance constrained requirements and function in a temporal distance-constrained manner, rather than just periodically. Such a model is defined by Hun, Lin et. al. in [20, 17]. In this model, the temporal distance between any two adjacent instances of the same task is bounded by some given upper bound. Scheduling schemes, schedulability conditions, and fundamental properties are given. Modifications of algorithms designed for periodic tasks systems and for the pinwheel problem to schedule distance constrained tasks are presented.

Examples of special cases of relative timing constraints are task couplings as considered e.g. in [15, 16, 25], where in any feasible schedule an upper bound gap, a lower bound gap, or an exact gap between jobs of specified pairs are required. Krings et al. [22] also consider coupled jobs where an upper bound for the gap is specified.

In this paper, we study a problem for distance constrained tasks on a single processor, and concentrate on the algorithms that minimize processor utilization. We show that the processor utilization can be reduced considerably while the response times are still within their limits; hence, the processor has more free capacity to process sporadic tasks. As a side effect, the way the tasks get scheduled to obtain minimal processor utilization results in jitter reduction improving system stability.

The paper is organized as follows: The following section introduces the model, specifies notation, and formulates the problem. Next, we present closely related models and transfer complexity issues. Then, simulation results are presented showing that processor utilization in the distance constrained model tends to be smaller than in the corresponding periodic model. As a theoretical result, exploring the border to intractability, we show a polynomial time solution for the two-task problem with zero release times that minimizes processor utilization. The last section addresses the summary and outlook on further problems left open for future research or being subject of work in progress.

## 2. Model and Problem Formulation

We consider the problem of non-preemptively scheduling sets of tasks on a single processor, each task representing an unbounded chain of task instances (or jobs) with relative release times and deadlines.

### Distance Constrained Task Model

Given a set of tasks  $T = \{T_1, \dots, T_n\}$ , we assume that they are independent of each other. Each task  $T_j$  ( $1 \leq j \leq n$ ) is characterized by a triple of parameters,  $(p_j, r_j, d_j)$ : processing time  $p_j$ , release time  $r_j$ , and deadline  $d_j$ . For a given task we assume that all its instances have the same processing time and relative timing constraints. A task  $T_j$  is the representative of a job sequence of unbounded length,  $T_j = (T_j^1, T_j^2, \dots)$ , where the jobs are chain-dependent ( $T_j^1 \prec T_j^2 \prec \dots$ ). We use the same notation for jobs as for tasks, but enhanced by the instance number. We say that two task instances  $T_j^i$  and  $T_j^{i+1}$  are coupled by relative timing constraints (relative release time and relative deadline) in the following sense: if  $T_j^i$  completes at time  $c_j^i$ , then the next instance,  $T_j^{i+1}$ , must not start before time  $c_j^i + r_j$ , and complete not later than  $c_j^i + d_j$ . Observe that  $d_j \geq r_j + p_j$  for feasibility reasons. The values  $r_j$  and  $d_j$  are referred to as *relative release time* and *relative deadline*, respectively.

### Problem Formulation

We assume that the problem of scheduling a given set of tasks is deterministic, i.e. that all timing parameters (processing times, relative release times and deadlines) are known and specified in ad-

vance, so that a schedule can be constructed off-line. The objective is to schedule the tasks non-preemptively on a single processor. Without loss of generality it is assumed that all time parameters are integers. For convenience reasons we omit the word "relative" and use the terms release time and deadline instead of relative release time and relative deadline, respectively. To define a reference point (offset) for the release times and deadlines of the first job  $T_j^1$  of each task  $T_j$ , we assume that it is set to 0.

A non-preemptive schedule for a set of tasks  $T = \{T_1, \dots, T_n\}$  on a single processor can be described by a sequence of jobs and idle intervals. For example, the sequence  $(T_j^1, I_k, T_j^2, I_k, \dots)$  represents the schedule where the first instance  $T_j^1$  of task  $T_j$  is followed by an idle interval (idle "instance") of length  $k$ , then followed by the second instance  $T_j^2$ , etc. To simplify notation we just specify it by the corresponding sequence of tasks with idle intervals  $(T_j, I_k, T_j, I_k, \dots)$ . Since each task has an unbounded number of jobs, the schedules are of infinite length. On the other hand, if a feasible schedule for a given set of tasks exists, there will also be one with cyclic behavior that can be described by a job sequence of finite length (cf. [FC02], Th. 2.1; [10], Th. 27.1). A partial schedule is referred to as a sequence of jobs (and idle intervals between them) of finite length. The concatenation of partial schedules  $S_1$  and  $S_2$  is denoted by  $S_1S_2$ . If  $S$  is a partial schedule,  $S,-$  denotes the schedule obtained by infinite repetition of  $S$ . We say that  $S$  is a generating sequence of  $S,-$ . To check the feasibility of a schedule it is easy to see that it suffices to check the feasibility of a partial schedule. In this paper, we consider the following problem: If a feasible solution exists, find a schedule that minimizes processor utilization.

### 3. Related Models

In this section, we briefly describe the relation of the DC model with closely related scheduling problems.

#### Comparison with Periodic Task Model

If we consider distance constrained (DC) tasks in a periodic paradigm, the period must be chosen such that the time lags between succeeding instances of the task never exceed  $d_j - p_j$ . The period  $\pi_j$  in a corresponding task system then calculates to  $\pi_j = (p_j + d_j)/2$ . This follows from the fact that it may happen that one instance is processed at the beginning of a period, and the next one at the end of the following interval. When comparing the processor utilization by task  $T_j$  we get: in the periodic task model:  $p_j/\pi_j = 2p_j/(p_j + d_j)$ , and in the DC model:  $p_j/d_j$ , assuming maximum gaps  $p_j - d_j$  between the instances.

Comparing the processor loads we see that in the periodic model the processor utilization is up to  $2d_j/(p_j + d_j)$  times larger than in DC model, or almost a factor of 2 more in case of small processing times. The following example illustrates this fact. Table 1 specifies five tasks with processing times and distance constraints. The last line shows the periods as calculated from the equation for  $\pi_j$ .

	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$
$p_j$	5	3	7	4	8
$d_j$	93	77	55	42	44
$\pi_j$	49	40	31	23	26

**Table 1.** An example task set with distance constraints and corresponding periods

In the periodic model, the processor utilization is  $U^{per} := \sum p_j / \pi_j = 0.88$ . Notice that neither a preemptive nor a non-preemptive periodic schedule exists. In the DC model,  $\sum p_j / d_j = 0.50$  is the lower bound for the utilization  $U^{DC}$  and, hence, named also in the literature the total density of task set.

An example of a feasible non-preemptive schedule with DC is  $(T_4 T_5 T_3 T_2 I_{20} T_5 T_3 T_2 T_1 I_{15}, \dots)$  (executed cyclically), following notations of

the section 2, with  $U^{rel} = 0.58$ . It shows that replacing the DC model by a periodic model can only be done at the cost of processor utilization, or it may even lead to a non-feasible problem. On the other hand, the periodic task model can always be replaced by a relative timing model by setting  $d_j := 2\pi_j - p_j$ .

Lower utilization demand is an important issue in practice if sporadic tasks need to be included. It would therefore be important to have sufficiently free utilization capacity available. Simulation results are presented in Figure 1. They show mean free processor capacity of the DC model (bold line) in comparison with the periodic model (dotted line). It can be seen that the processor utilization of relative timing schedules is smaller than in a corresponding periodic model.

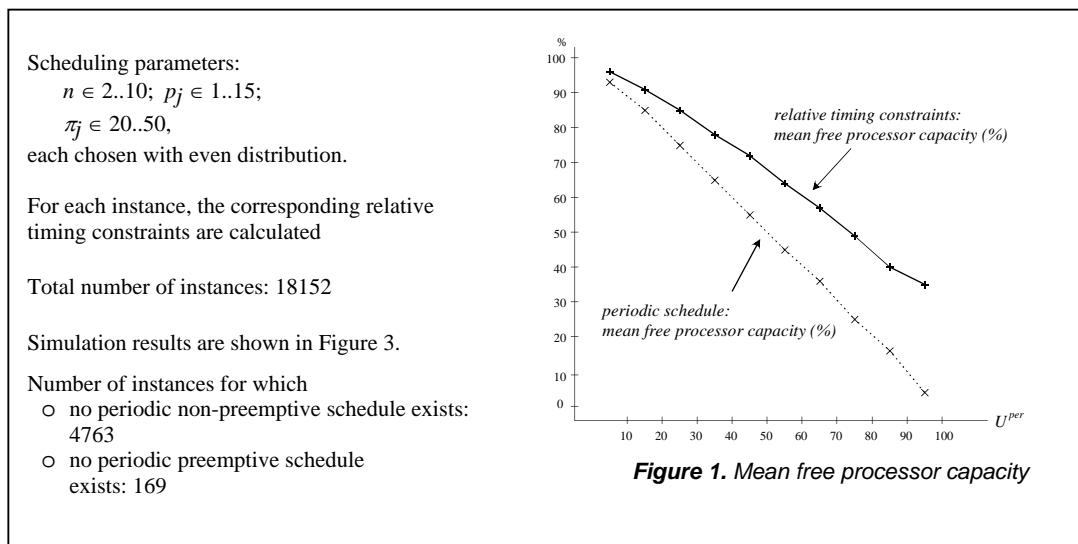
### Relationship with Pinwheel Problems

The well known *Pinwheel Problem* (PP) [21] is closely related to the DC model, in the sense that every instance of PP can easily transformed into an instance of the DC model by setting specific deadlines, zero release times, and unit processing times. But in the PP the processor utilization is always dense. Several algorithms for scheduling of pinwheel instances are devised in [8, 7]. Associated real time problems, properties of PP and relationship with other models are discussed in [10]. The pinwheel is viewed as a special case of periodic tasks scheduling problem, the discrete version of the distance constrained tasks system problem, proportional fairness, periodic maintenance problems and others. The *Generalized Pinwheel Problem* (GPP), introduced by Feinberg and Curry [12], is based essentially on the same model as DC, so results concerning complexity etc. can be transferred. The problem does not aim on minimization of utilization, though.

### Complexity Considerations

In [11] it is shown that the Generalized Pinwheel Problem is NP-hard. Since GPP is based essentially on the same model as DC and constitutes a subclass, we can conclude that the DC scheduling problem is also NP-hard.

**Theorem:** *The problem of finding a feasible schedule in the distance constrained task model is NP-hard.*



### 3. Minimizing Processor Utilization

We now turn to the optimization problem to find a schedule with minimum processor utilization. We give an answer to this question for the simple case of two tasks  $T_1$  and  $T_2$  with zero release times.

Notice that, as long there are no sporadic tasks, and only feasibility is concerned, there is no need to include idle intervals on the processor. The only condition for schedulability is that an instance of  $T_2$  can be included between two instances of  $T_1$ , and vice versa. Hence, the following is trivial:

**Lemma 1.** Tasks  $T_1$  and  $T_2$  with zero release times can be feasibly scheduled iff  $p_1 + p_2 \leq \min\{d_1, d_2\}$ .  $\square$

In the following, we assume that there exists a feasible schedule for  $\{T_1, T_2\}$ . Let w.l.o.g. be  $d_1 \leq d_2$ , and define  $p = p_1 + p_2$ .

Since we can restrict to cyclic schedules it is sufficient to consider partial schedules  $S$  of finite lengths, for which  $S, \bar{\phantom{S}}$ , is feasible. Let  $C_S$  be the makespan of  $S$ ; if  $S$  has  $n_j$  instances of  $T_j$  ( $j = 1, 2$ ), then the utilization within the time span  $C_S$  is  $U = (n_1p_1 + n_2p_2)/C_S$ . Even  $S$  is repeated infinitely often, the utilization of  $S, \bar{\phantom{S}}$ , is  $U$ .

A general guideline for minimizing utilization would be to try to schedule the instances of each task at maximum possible distances, i.e.  $d_j - p_j$  for  $T_j$ ,  $j = 1, 2$ . If the task instances can be scheduled at these distances, the utilization of  $S$  is  $U = p_1/d_1 + p_2/d_2$ .

**Lemma 2.** The tasks  $T_1$  and  $T_2$  with zero release times can be scheduled at distances  $d_1 - p_1$  for  $T_1$ , and  $d_2 - p_2$  for  $T_2$  without conflict if and only if  $p \leq \gcd(d_1, d_2)$ , where  $\gcd$  is the greatest common divisor.

*Proof.* Let  $S^j$  be a schedule for the  $T_j$  only, where the first instance of  $T_j$  starts at time 0, and the following jobs are scheduled at maximal possible distances; then at each time  $id_j$  an  $i$ -instance of  $T_j$  is started. The question is whether both schedules,  $S^1$  and  $S^2$ , can be merged into one schedule in such a way that the distances between the jobs are kept, and no overlap occurs.

It is a well-known fact that, given two integers  $d_1$  and  $d_2$ , we can always find integers  $k_1$  and  $k_2$  such that  $k_2d_2 - k_1d_1 = \gcd(d_1, d_2)$ . It is clear that  $s = \gcd(d_1, d_2)$  is the smallest distance  $> 0$  between start times of jobs in  $S^1$  and  $S^2$ . A merged schedule,  $S$ , is constructed by scheduling the jobs of  $T_1$  at times  $id_1$  and those of  $T_2$  at times  $id_2 + p_1$  ( $i = 0, 1, \dots$ ). Let integers  $k_1$  and  $k_2$  be such that  $k_2d_2 - k_1d_1 = s$ . Since the  $(k_1+1)^{\text{st}}$  job of  $T_1$  starts at  $k_1d_1$ , the  $(k_2+1)^{\text{st}}$  job of  $T_2$  starts at time  $k_1d_1 + p_1$  and completes at time  $k_1d_1 + p_1 + p_2 \leq k_1d_1 + s = k_2d_2$ . Hence the tasks can be scheduled without overlap in  $S$  if and only if  $p = p_1 + p_2 < s$ .  $\square$

If  $p_1 + p_2 > \gcd(d_1, d_2)$ , we see that the maximal possible distances cannot be kept because of conflicts. In the following we analyze this situation in detail.

We may assume that the schedule  $S$  has jobs  $T_1^k, T_2^l$  scheduled such that there is no idle interval in between. If there is no such a pair we can shift the schedule from the second job on to the left until there is no gap between them. With this assumption, it is easy to see that there are three types of cycles (types I, II, and III), as depicted in Figures 6 – 8. Types I and II differ by the task that is selected first. Type III is a combination of types I and II.

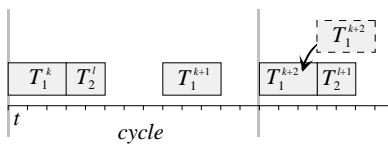


Figure 2 Cycle of type I

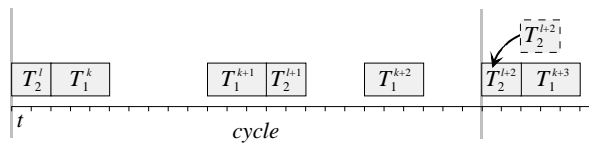


Figure 3 Cycle of type II

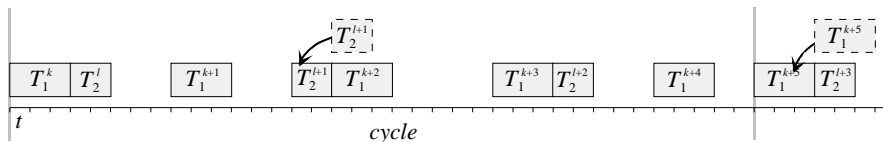


Figure 4 Cycle of type III

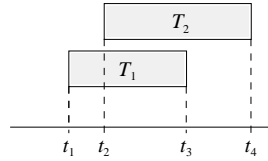
Let  $n^I_{,1}$ ,  $n^II_{,1}$ ,  $n^III_{,1}$ ,  $n^I_{,2}$ ,  $n^II_{,2}$ ,  $n^III_{,2}$  be the respective maximal numbers of  $T_1$  and  $T_2$  jobs in the cycles of type I, II and III at maximum possible distances as shown in Figures 6 – 8. Our aim is to determine for each cycle type the smallest values for these numbers.

**Lemma 3.** (i)  $n^I_{,2}d_2 < n^I_{,1}d_1 < n^I_{,2}d_2 + p$ ;

(ii)  $n^II_{,1}d_1 < n^II_{,2}d_2 < n^II_{,1}d_1 + p$ .

*Proof.* (i) follows directly from the conditions for building type-I cycles; (ii) is symmetric.

We show this for type II cycles: let  $t_1$  and  $t_3$  be the respective start and completion time of the first instance of  $T_1$  that overlaps with the instance of  $T_2$ :  $t_1 = t + n^II_{,1}d_1 + p_2$ ,  $t_3 = t + n^II_{,1}d_1 + p$  (where  $t$  is the start time of the cycle). The overlapping starts at time  $t_2 = t + n^II_{,2}d_2$  and completes at time  $t_4 = t + n^II_{,2}d_2 + p_2$ . Thus the condition for an overlap is:  $t_3 > t_2$  and  $t_1 < t_4$  (see Figure 9), i.e.  $n^II_{,1}d_1 < n^II_{,2}d_2 < n^II_{,1}d_1 + p$ .  $\square$



**Figure 5** Overlapping task instances

**Cycle of type I:** This cycle starts with an instance of  $T_1$ , followed immediately by an instance of  $T_2$ . The conflict defining the end of the cycle is resolved by moving the instance of  $T_1$  in front of the instance of  $T_2$  (see Figure 2). The start time of the  $T_1$  instance then defines the end of the cycle.

**Lemma 4.**  $n^I_{,1}$  is the smallest natural number for which  $n^I_{,1}d_1/d_2 - \lfloor n^I_{,1}d_1/d_2 \rfloor < p/d_2$ , and  $n^I_{,2} = \lfloor n^I_{,1}d_1/d_2 \rfloor$ .<sup>1</sup> The cycle of type I has length  $C^I = n^I_{,2}d_2$ , and the processor utilization is  $U^I = (n^I_{,1}p_1 + n^I_{,2}p_2)/C^I$ .

*Proof.* Let  $t$  be the time when the cycle starts. Since  $T_2$  keeps its maximum distances, the obtained cycle lasts from  $t$  to  $t + n^I_{,2}d_2$ , and we get

$$C^I = n^I_{,2}d_2 \quad (1)$$

for the length of the type I cycle. Let  $\rho = p/d_2$  and  $\varepsilon = d_1/d_2$ , then  $n^I_{,2} < n^I_{,1}\varepsilon < n^I_{,2} + \rho$  from Lemma 3 (i).

(i). Separating the integer part of  $n^I_{,1}\varepsilon$ , i.e.,  $n^I_{,1}\varepsilon = \lfloor n^I_{,1}\varepsilon \rfloor + \alpha$ , we get  $n^I_{,2} < \lfloor n^I_{,1}\varepsilon \rfloor + \alpha < n^I_{,2} + \rho$ , which, since  $0 < \alpha, \rho < 1$ , is only possible if  $n^I_{,2} = \lfloor n^I_{,1}\varepsilon \rfloor$  and  $\alpha = n^I_{,1}\varepsilon - \lfloor n^I_{,1}\varepsilon \rfloor < \rho$ .

The minimally condition for  $n^I_{,1}$  leads to the result that  $n^I_{,1} > 0$  is the smallest integer for which  $n^I_{,1}\varepsilon - \lfloor n^I_{,1}\varepsilon \rfloor < \rho$ .

The processor utilization  $U^I$  of this cycle is given by

$$U^I = (n^I_{,1}p_1 + n^I_{,2}p_2)/C^I. \quad \square$$

The other possibility of resolving the conflict, i.e., shifting the  $T_2$ -instance in front of the  $T_1$ -instance, leads to a cycle of type III. This is discussed below (see *cycle of type III*).

**Cycle of type II:** This cycle starts with a  $T_2$ -instance, followed immediately by a  $T_1$ -instance. The conflict defining the end of the cycle is resolved by moving the  $T_2$ -instance in front of the  $T_1$ -instance (see Figure 3). The start time of the  $T_2$ -instance then defines the end of the cycle.

**Lemma 5.**  $n^II_{,1}$  is the smallest natural number for which  $n^II_{,1}d_1/d_2 - \lfloor n^II_{,1}d_1/d_2 \rfloor > 1 - p/d_2$ , and  $n^II_{,2} = \lfloor n^II_{,1}d_1/d_2 \rfloor + 1$ . The length  $C^{II}$  of the cycle is  $n^II_{,1}d_1$ , and the processor utilization is  $U^{II} = (n^II_{,1}p_1 + n^II_{,2}p_2)/C^{II}$ .

*Proof.* As compared to the cycle of type I, the roles of  $T_1$  and  $T_2$  are now interchanged. Hence the length  $C^{II}$  of the type-II cycle is

<sup>1</sup>  $\lfloor a \rfloor$  denotes the largest integer  $\leq a$ .

$$C^{\text{II}} = n^{\text{II}}_{,1} d_1, \quad (2)$$

and from Lemma 3 (ii) we get ( $\varepsilon$  and  $\rho$  are defined as in the proof of Lemma 4)

$$n^{\text{II}}_{,1} \varepsilon < n^{\text{II}}_{,2} < n^{\text{II}}_{,1} \varepsilon + \rho.$$

Define  $\beta = n^{\text{II}}_{,1} \varepsilon - \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor \in (0, 1)$ , then

$$\lfloor n^{\text{II}}_{,1} \varepsilon \rfloor < n^{\text{II}}_{,2} - \beta < \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor + \rho = \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor + 1 - (1 - \rho),$$

which is only possible if  $\beta = n^{\text{II}}_{,1} \varepsilon - \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor > 1 - \rho$  and  $n^{\text{II}}_{,2} = \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor + 1$ . Hence from the minimally condition we get that  $n^{\text{II}}_{,1}$  is the smallest natural number for which  $\beta = n^{\text{II}}_{,1} \varepsilon - \lfloor n^{\text{II}}_{,1} \varepsilon \rfloor > 1 - \rho$ .

The processor utilization  $U^{\text{II}}$  of this cycle is given by

$$U^{\text{II}} = (n^{\text{II}}_{,1} p_1 + n^{\text{II}}_{,2} p_2) / C^{\text{II}}. \quad \blacksquare$$

The other possibility of resolving the conflict, i.e., shifting the  $T_1$ -instance in front of the  $T_2$ -instance, is discussed next.

**Cycle of type III:** A cycle of type III is constructed by combining cycles of type I and II (see Figure 4): Starting with a cycle of type I, we shift the conflicting  $T_2$ -instance in front of the  $T_1$ -instance, which results in the initial condition of the type II cycle. The conflict at the end of the cycle is resolved by shifting the  $T_1$ -instance in front of the  $T_2$ -instance. This leads to the initial configuration of the type-I cycle. The cycle length  $C^{\text{III}}$  can then easily be determined as

$$C^{\text{III}} = n^{\text{I}}_{,1} d_1 + n^{\text{II}}_{,2} d_2 - p$$

Notice that the number of  $T_1$ - and  $T_2$ -instances in this cycle is  $n^{\text{III}}_{,1} = n^{\text{I}}_{,1} + n^{\text{II}}_{,1}$ ,  $n^{\text{III}}_{,2} = n^{\text{I}}_{,2} + n^{\text{II}}_{,2}$ , respectively. Hence the processor utilization is  $U^{\text{III}} = [(n^{\text{I}}_{,1} + n^{\text{II}}_{,1})p_1 + (n^{\text{I}}_{,2} + n^{\text{II}}_{,2})p_2] / (n^{\text{I}}_{,1} d_1 + n^{\text{II}}_{,2} d_2 - p)$ .

**Comparing utilizations:** It turns out that, for particular pairs of tasks, any of the cycles may have minimum processor utilization. However, it can be shown that the cases I or II outperform case III if  $U^{\text{I}}$  and  $U^{\text{II}}$  are not too close.

**Theorem 1.** *Given tasks  $T_1$  and  $T_2$  with zero release times, and  $p_1 + p_2 > \gcd(d_1, d_2)$ . If  $T_1$  and  $T_2$  can be feasibly scheduled then the schedules  $S^{\text{I}}$ ,  $S^{\text{II}}$ , and  $S^{\text{III}}$  obtained by executing the respective cycles of type I, II, III are feasible. The optimal schedule can be found among these schedules, and the respective processor utilizations are*

$$\begin{aligned} U^{\text{I}} &= (n^{\text{I}}_{,1} p_1 + n^{\text{I}}_{,2} p_2) / C^{\text{I}} \\ U^{\text{II}} &= (n^{\text{II}}_{,1} p_1 + n^{\text{II}}_{,2} p_2) / C^{\text{II}} \\ U^{\text{III}} &= [(n^{\text{I}}_{,1} + n^{\text{II}}_{,1}) p_1 + (n^{\text{I}}_{,2} + n^{\text{II}}_{,2}) p_2] / C^{\text{III}}. \end{aligned} \quad (3)$$

Furthermore, (i) if  $U^{\text{II}} \geq U^{\text{I}}(1 + p/C^{\text{II}})$  then  $U^{\text{I}} < U^{\text{III}} < U^{\text{II}}$ , and (ii) if  $U^{\text{I}} \geq U^{\text{II}}(1 + p/C^{\text{I}})$  then  $U^{\text{II}} < U^{\text{III}} < U^{\text{I}}$ .

*Proof.* Schedule  $S^{\text{I}}$  (and similarly  $S^{\text{II}}$  and  $S^{\text{III}}$ ) is obtained by repeated execution of the cycle of type I. The feasibility of  $S^{\text{I}}$ ,  $S^{\text{II}}$  and  $S^{\text{III}}$  follows from the condition  $p_1 + p_2 \leq \min\{d_1, d_2\}$  and the above discussion of the cycle types.

From Lemma 3, and equ. (1), (2) we get  $C^{\text{I}} + C^{\text{II}} - p < C^{\text{III}} < C^{\text{I}} + C^{\text{II}} + p$ . With this, (i) and (ii) follow directly from (3).  $\blacksquare$

## 4. Summary and Outlook

In this paper we presented results of our ongoing work on a topic in hard real time systems. We analyzed the problem of scheduling unbounded sequences of tasks with relative timing constraints subject to minimizing processor utilization. The presented problem turned out to be NP-hard in general, but we gave an in-depth analysis of the two-task case that was shown to be solvable in polynomial time.

Many problems remain to be worked on in future research.

The introduced distance constrained (DC) model comprises the classical periodic model in the sense that the periodic parameters can be transformed to distance constrained conditions. More-

over, the mean processor utilization tends to decrease while changing to the DC model. Most commercial real time system kernels, though, are based on priority schemes. Usually they feature only few fixed priority levels, an implementation of the DC model on top of those kernels is neither easy nor efficient, cf. [1]. But integrating the DC model into a real time system kernel is subject of ongoing development.

As a consequence of minimizing processor utilization, free processor capacity remains to process aperiodic tasks. But at arrival of aperiodic tasks, the system has to decide what to do depending on a schedulability analysis. Different mechanisms, algorithms and complexity issues concerning this schedulability analysis are also of interest in further research. Moreover, what is the maximal allowed size and frequency of a sporadic task to be accepted?

Since there is no necessary and sufficient condition that can be checked in reasonable time, unless  $P=NP$ , the question about heuristic approaches and polynomial solvable subclasses is obvious. Also the exploration of the border to intractability is of interest. E.g. what is the time complexity of the three-task case? Is there still a polynomial solution? Some of these topics are already answered or dealt with, respectively, in a comprehensive theoretical framework currently developed.

Finally, we summarize some of the advantages of the relative timing approach as compared to the periodic approach:

- (a) Periodic requirements can always be replaced by relative timing constraints; in this sense the relative timing approach is more general.
- (b) Generally, a relative timing schedule allows including longer sporadic tasks than a corresponding periodic schedule.
- (c) If an off-line schedule is required, the periodic model may lead to very long schedules where the schedule length is defined by the least common multiple of the period lengths. In contrast, the relative timing model leads in average to considerably shorter schedules.

## References

- [1] G. Buttazzo (2005), Rate Monotonic vs. EDF: Judgment Day, *Real-Time Systems* **29**(1), 5 – 26.
- [2] E. Bini, G. Buttazzo (2003), Rate Monotonic Analysis: The Hyperbolic Bound, *IEEE Transactions on Computers* **52**(7), 933 – 942.
- [3] S.-T. Cheng, A. K. Agrawala (1995), Allocation and scheduling of real-time periodic tasks with relative timing constraints, *2nd International Workshop on Real-Time Computing Systems and Applications*, 210 – 217.
- [4] S. Choi, A. K. Agrawala (1997), Scheduling aperiodic and sporadic tasks in hard real-time systems, Report CS-TR-3794, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland.
- [5] S. Choi, A. K. Agrawala (2000), Dynamic dispatching of cyclic real-time tasks with relative timing constraints, *Real-Time Systems* **19**(1), 5 – 40.
- [6] S. Choi, A. K. Agrawala, L. Shi (1997), Designing dynamic temporal controls for critical systems, Report CS-TR-3804, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland.
- [7] M.Y. Chan, F. Chin (1992), General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction, *IEEE Transactions on Computers* **41**(6), 755 – 768.
- [8] M.Y. Chan, F. Chin (1993), Schedulers for Larger Classes of Pinwheel Instances, *Algorithmica* **9**, 425 – 462.
- [9] S. Choi (1997), Dynamic Time-Based Scheduling for Hard Real-Time Systems. Ph.D. Thesis, University of Maryland.
- [10] D. Chen, A. Mok. (2004), The Pinwheel: A Real-Time Scheduling Problem. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRT Press, chapter 27, 1 – 17.
- [11] E.A. Feinberg, M.A. Bender, M.T. Curry, D. Huang, T. Koutsoudis, J.L. Bernstein (2002), Sensor resource management for an airborne early warning radar, *Proceedings of SPIE, Signal and Data Processing of Small Targets*, Vol. 4728, 145 – 156, August 2002.
- [12] E.A. Feinberg, M.T. Curry (2005), Generalized Pinwheel Problem, *Mathematical Methods of Operations Research* **62**, 99 – 122.

- [13] R. Gerber (1995), 'Guaranteeing end-to-end timing processes', *Proc. IEEE Real-Time Systems*, 192 – 203.
- [14] R. Gerber, S. Hong, M. Saksena (1995), 'Guaranteeing real-time requirements with resource-based calibration of periodic processes', *IEEE Transactions on Software Engineering* **21**( 7), 579 – 592.
- [15] R. Gerber, W. Pugh, M. Saksena (1995), 'Parametric dispatching of hard real-time jobs', *IEEE Transactions on Computers* **44**(3).
- [16] J.N.D. Gupta (1996), 'Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per task and time lags', *Journal of Global Optimization* **9**, 239 – 253.
- [17] C.-W. Hsueh and K.-J. Lin (2001), 'Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model', *IEEE Transactions on Computers* **50**, 51 – 66.
- [18] C.-C. Han, K.J. Lin (1989), 'Scheduling jobs with Temporal Consistency Constraints', *Proc. Sixth IEEE Workshop Real Time Operating Systems and Software*, 18 – 23, Pittsburgh.
- [19] C.-C. Han, K.J. Lin (1992), 'Scheduling distance-constrained real-time tasks', *IEEE Real-Time Systems Symposium*, 300 – 308.
- [20] C.-C. Han, K.J. Lin, C.-J. Hou (1996), 'Distance-constrained Scheduling and Its Application to Real-Time Systems', *IEEE Transactions on Computers* **45**, 814 – 826.
- [21] R. Holte, L. Rosier, I. Tulchinsky, D. Varvel (1992), 'Pinwheel scheduling with two distinct numbers', *Theoretical Computer Science* **100**, 105 – 135.
- [22] A. W. Krings, M. H. Azadmanesh (1999), 'Avoiding run-time infeasibility in systems containing coupled tasks', *INFOR (Information Systems & Operational Research) Journal* **37**(1), 77 – 88.
- [23] C.L. Liu, J. W. Layland (1973), 'Scheduling algorithms for multiprogramming in a hard-real-time environment', *J. ACM* **20**(1), 46 – 61.
- [24] A.K. Mok, D. Chen (1997), 'A multiframe model for real-time tasks', *IEEE Transaction on Software Engineering* **23**(10), 635 – 645.
- [25] A.J. Orman, C.N. Potts (1997), 'On the complexity of coupled-job scheduling', *Discrete Applied Mathematics* **72**, 141 – 154.
- [26] M. Spuri, G. Buttazzo (1996), 'Scheduling Aperiodic Tasks in Dynamic Priority Systems', *The Journal of Real-Time Systems* **10**(2), 179 – 210.