Feasibility Problems for Recurring Tasks on One Processor*

Sanjoy K. Baruah
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
U.S.A.

Rodney R. Howell
Dept. of Computing and Information Sciences
Kansas State University
Manhattan, KS 66506
U.S.A

Louis E. Rosier†
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
U.S.A.

March 18, 1992

Abstract

We give a comprehensive summary of our recent research on the feasibility problems for various types of hard-real-time preemptive task systems on one processor. We include results on periodic, sporadic, and hybrid task systems. While many of the results herein have appeared elsewhere, this is the first paper presenting a holistic view of the entire problem.

1 Introduction

Scheduling theory as it applies to hard-real-time environments has lately become a topic of much interest. In a hard-real-time environment, the missing of a single deadline may have disastrous consequences; hence, in such an environment, it is required that all deadlines be met. The tasks in such an environment are often of a recurring nature. For example, in a *periodic* task system [LL73, LM80, LM81, LW82, Mok83, BHR90], each task makes a request for the processor at regular periodic intervals. Each time a request is made, a certain amount of processing time must be allocated to that task before its deadline expires. The processing time and the time elapsed between the request and the deadline are always the same for each request of a particular task; however, these times may be different for different tasks. Preemptions are allowed at integer time values[1] without penalty. A *sporadic* task system [Mok83, LSS87, HL88, SLS88, JSM91, SSL89, BMR90] is similar, except that the request times cannot be predicted; thus, sporadic task systems may be used to model

*This work was supported in part by National Science Foundation Grant No. CCR-8711579. Some of these results were originally presented at the *15th International Symposium on Mathematical Foundations of Computer Science*, Banská Bystrica, Czechoslovakia, August 1990, and the *11th Real-Time Systems Symposium*, Lake Buena Vista, Florida, U.S.A., December 1990.

†Louis Rosier passed away on May 6, 1991, while this paper was under review.

[1] All of the results herein may be extended to allow noninteger preemptions; see [BHR90].

event-driven systems. We do, however, require a minimum separation time between successive requests of the same task. We also consider *hybrid* task systems composed of both periodic tasks and sporadic tasks.

The ultimate goal regarding real-time task systems is to find an offline algorithm that will mechanically synthesize online algorithms for scheduling the processor. The offline algorithm would first determine the *feasibility* of a task system. A task system is said to be *feasible* if any legal set of requests has a corresponding schedule in which no deadlines are missed. If the task system is feasible, the offline algorithm would then construct a suitable online algorithm for scheduling it. Feasibility is a necessary and sufficient condition for the *deadline algorithm* (see, e.g., [LL73, Lab74, Der74]) to be guaranteed to produce a valid schedule (i.e., one in which no deadlines are missed) for any legal set of requests. The deadline algorithm therefore constitutes a suitable online scheduling algorithm whenever one exists. If we could couple the deadline algorithm with a polynomial time or even a pseudo-polynomial time feasibility test, we would have made significant progress toward the realization of a viable algorithm for mechanical synthesis.

The feasibility problem for periodic task systems was examined in [LM80] and later in [BHR90]; for sporadic systems, it was examined in [BMR90]. All of the algorithms derived are in some way based upon the deadline algorithm. The purpose of this paper is to summarize the results from [BHR90] and [BMR90], giving a unified approach to all of the problems. In the process, we are able to streamline many of the proofs and to present a comprehensive view of the feasibility problems for various types of recurring task systems. Although most of these results can be found in either [BHR90] or [BMR90], we do generalize the results on periodic systems to allow the deadlines to exceed the periods. Furthermore, we extend the results of [BHR90, BMR90] to hybrid systems.

We begin in Section 2 by defining task systems in a very general framework. Periodic, sporadic, and hybrid task systems are then special cases of a task system. Furthermore, the deadline algorithm remains optimal for this very general definition — that is, it produces a valid schedule whenever one exists. We then give as our main result of Section 2 a necessary and sufficient condition for a task system to be feasible. This condition is a generalization of some of the techniques given in [BHR90, BMR90]. Whenever a task system is not feasible, this condition states the existence of a set of requests and a time interval such that the processor time required by that set of requests within that interval exceeds the time available in the interval. Thus, in order to show a task system to be infeasible, we need only to consider a function, defined for each set of requests, giving the amount of processing time required within each time interval. By finding the appropriate set of requests and the appropriate time interval, simply evaluating the function will show the task system to be infeasible. What remains to be shown for each class of task systems is an upper bound on the cost of finding the appropriate set of requests and the appropriate time interval, and of computing the processing time required within that interval.

In Section 3, we consider the feasibility problem for periodic task systems. In [LM80], this problem was shown to be co-NP-hard in the ordinary sense; however, no upper bound better than PSPACE was given. We first consider the upper bound. For a periodic task system, there is only one legal set of requests. Thus, our main result from Section 2 implies that we only need to show how to find the appropriate time interval, and how to compute the amount of processing time demanded within it. We address both of these problems, showing that the time interval may be nondeterministically guessed in polynomial time, and the processing time required may be computed in polynomial time. Hence, the feasiblity problem is shown to be in co-NP. Using the framework introduced in Section 2, as well as techniques developed in [BMR90], we eliminate much of the dependency on the earlier work of Leung and Merrill [LM80] that was present in [BHR90]; furthermore, the proofs are streamlined significantly. We conclude the section by showing the feasibility problem for periodic task systems to be co-NP-complete in the strong sense, even for a very restricted subclass.

In Section 4, we consider the feasibility problem for sporadic task systems. We first show that this problem is equivalent to the feasibility problem for synchronous task systems — those periodic task systems in which all start times are zero. The upper bounds given in Section 3 for periodic task systems therefore follow immediately for sporadic task systems. Furthermore, we go on to show that for a special case that makes up the vast majority of sporadic (or synchronous) task systems, the feasibility problem can be solved in pseudo-polynomial time. This result may be contrasted with a result from Section 3, which states that the feasibility problem for periodic task systems is co-NP-hard in the strong sense, even if the periodic systems are restricted in the same way. As in Section 3, these proofs are more self-contained and streamlined than those in [BHR90, BMR90]. At this time, we do not know whether, in general, the feasibility problem for sporadic (synchronous) task systems can be solved in pseudo-polynomial time, or perhaps even polynomial time.

In Section 5, we consider the feasibility problem for hybrid task sytems. Since hybrid task systems are a generalization of periodic task systems, the co-NP-hard (in the strong sense) lower bound of Section 3 applies to hybrid task systems. Still, even though hybrid task systems are more general than periodic task systems, we are able to make use of the upper bounds of Section 3 in dealing with hybrid task systems. One can envision a set of periodic task systems, each of which behaves as one of the possible behaviors of the given hybrid task system (though clearly we cannot mimic all of the possible behaviors of the hybrid task system in this way). We are able to give a succinct proof that if the hybrid task system is infeasible, then one of these periodic task systems must be infeasible; furthermore, it is possible to nondeterministically guess the appropriate periodic task system in polynomial time. Therefore, from the results of Section 3, the feasibility problem for hybrid task systems is co-NP-complete in the strong sense.

We conclude in Section 6 with a discussion of the remaining open problems and of the impact of our results on other problems. In particular, in order to show, in Section 3, that the feasibility problem for periodic task sytems is co-NP-hard in the strong sense, we first show the Simultaneous Congruences Problem [LW82] to be NP-hard in the strong sense. It follows immediately from this last result that several problems shown to be co-NP-hard in [LW82, Leu89] are, in fact, co-NP-hard in the strong sense.

2 Task Systems

A *task system* is a set $\tau = \{T_1, ..., T_n\}$. The tasks T_i of τ may be defined in various ways depending upon what type of task system τ is (e.g., periodic, sporadic, or hybrid), but each T_i always contains in its definition an integer *execution time* $e_i \geq 1$ and an integer *deadline* $d_i \geq e_i$. Each task may make *requests* from time to time. Each request of task T_i is of the form (i, t), where $t \in N$ is the time at which the request was made. In the context of τ, we call (i, t) a τ*-request*. Each τ-request (i, t) requires the processor to be allocated to (i, t) for e_i time units during the interval $[t, t + d_i)$. These requirements allow preemptions at any integer time value without penalty. Let R be a (possibly infinite) set of τ-requests. R may or may not be *legal*; the conditions under which R is legal depend upon what type of task system τ is. R is *schedulable* iff the requirements of all τ-requests in R can be met. τ is *feasible* iff each legal set of τ-requests is schedulable.

In this paper, we discuss three types of task systems: periodic, sporadic, and hybrid. Each task T_i in a periodic task system is a 4-tuple, $T_i = (s_i, e_i, d_i, p_i) \in N^4$, where N denotes the natural numbers, s_i is the start time, $e_i \geq 1$ is the execution time, $d_i \geq e_i$ is the deadline, and $p_i \geq e_i$ is the period. A periodic task system has exactly one legal set of requests, $\{(i, t) \mid t = s_i + kp_i \text{ for some } k \in N\}$; for a periodic task system τ, we call this set of τ-requests R_τ. By applying the above definition of feasibility, we can see that τ is feasible iff R_τ is schedulable. Each task T_i in a sporadic task system is a triple $T_i = (e_i, d_i, p_i) \in N^3$, where $e_i \leq d_i$ are as above and p_i is the minimum separation time. A set R of requests for a sporadic task system is legal iff for each pair of distinct requests (i, t_1) and (i, t_2), $|t_1 - t_2| \geq p_i$. A hybrid task system contains both periodic tasks and sporadic tasks. Let $\tau = \{T_1, ..., T_n, T_{n+1}, ...T_{n+m}\}$, where $T_1, ..., T_n$ are periodic tasks and $T_{n+1}, ..., T_{n+m}$ are sporadic tasks. Each periodic task is a 4-tuple as in a periodic task system, and each sporadic task is a triple as in a sporadic task system. A set R of τ-requests is legal iff the following two conditions hold:

1. for all i, $1 \leq i \leq n$, $(i, t) \in R$ iff $t = s_i + kp_i$ for some $k \in N$; and
2. for all i, $n + 1 \leq i \leq m$, for each pair of distinct requests (i, t_1) and (i, t_2) in R, $|t_1 - t_2| \geq p_i$.

In studying the feasibility of task systems, the notion of an optimal scheduling algorithm is very helpful. A scheduling algorithm σ is said to be *optimal* for a certain class of task systems if, given any legal set of τ-requests R (where τ is in the respective class), σ produces a schedule that satisfies the requirements of all τ-requests in R whenever R is schedulable. For example, the deadline algorithm of Dertouzos [Der74], which we define below, is optimal for any class of task systems (i.e., no matter how the notion of a legal set of tasks requests is defined). Having an optimal scheduling algorithm allows us to avoid having to consider all possible ways of scheduling a given set of τ-requests; instead, we just consider the schedule produced by the deadline algorithm, knowing that if it is possible to meet all the requirements, that schedule will do so.

We will now describe the deadline algorithm, which we will denote by σ throughout the remainder of this paper. First, σ is an *online* algorithm; i.e., σ is an iterative algorithm which, when presented with a set of τ-requests R (τ-request (i, t_0) in R is presented to σ at time t_0), uniquely determines (in a manner described below) at each instant of time $t = 0, 1, 2, \ldots$, which *active* τ-request of R — if any — is to be allocated the processor. The τ-request (i, t_0) in R is said to be *active* at time t (with respect to σ) iff $t_0 \leq t < t_0 + d_i$ and σ did not allocate the processor to (i, t_0) for e_i time units in the interval $[t_0, t)$. σ reports failure — but does not terminate — at time t if there is a τ-request (i, t_0) in R such that $t_0 + d_i = t$ and σ did not allocate the processor to (i, t_0) for e_i time units in the interval $[t_0, t)$. σ can only successfully schedule τ if it never reports failure when presented with a legal set of τ-requests. Hence, σ can be expressed as a function of R and t:

$$(\sigma.R).(t) = \begin{cases} (i, t_0), \textit{failure} & \text{meaning that } \sigma \text{ at time } t \text{ allocates the processor to } \tau\text{-request } (i, t_0) \text{ and reports failure.} \\ (0, 0), \textit{failure} & \text{meaning that } \sigma \text{ at time } t \text{ leaves the processor idle and reports failure.} \\ (i, t_0) & \text{meaning that } \sigma \text{ at time } t \text{ allocates the processor to } \tau\text{-request } (i, t_0) \text{ and does not report failure.} \\ (0, 0) & \text{meaning that } \sigma \text{ at time } t \text{ leaves the processor idle and does not report failure.} \end{cases}$$

The method σ uses to determine which task is to be allocated the processor is as follows. At time t, σ allocates the processor to the active τ-request (if there is one) whose deadline is nearest. Ties can be broken in an arbitrary fashion without affecting whether or when $(\sigma.R)$ reports failure. Hence, without loss of generality we assume σ chooses active τ-request (i, t_1) over (j, t_2) whenever either $t_1 + d_i < t_2 + d_j$ or both $t_1 + d_i = t_2 + d_j$ and $i < j$. We now reproduce the following lemma from [Der74].

Lemma 2.1: (from [Der74]) σ is optimal for task systems; i.e., given a set of τ-requests R, $(\sigma.R)$ will construct a schedule for R if one exists; otherwise σ at some time will report failure.

Let $\tau = (T_1, ..., T_n)$ be a periodic, sporadic, or hybrid task system as defined above. We define the *processor utilization* of τ to be $\sum_{i=1}^{n} e_i/p_i$. Intuitively, the processor utilization gives the maximum fraction of processor time that can be demanded in any legal set of τ-requests. Clearly, in any feasible task system, the processor utilization can be no more than 1. As a shorthand convention, we will let $P = \text{lcm}\{p_1, ..., p_n\}$. Given a set of τ-requests R, we also define a function $g_R : N \times N \rightarrow N$ as follows:

$$g_R(t_1, t_2) = \sum_{(i,t) \in R(t_1, t_2)} e_i,$$

where $R(t_1, t_2) = \{(i, t) \mid t_1 \leq t \leq t_2 - d_i\}$. Thus, $g_R(t_1, t_2)$ gives the total amount of execution time required between t_1 and t_2. The following theorem and its corollary demonstrate the importance of the function g_R to the feasibility of a task system.

Theorem 2.1: Let $\tau = \{T_1, .., T_n\}$ be a task system. Suppose τ is not feasible, and let R be a legal set of τ-requests for which σ reports failure at the earliest time t_f. Then for some $t_b < t_f$, $g_R(t_b, t_f) > t_f - t_b$.

Proof: Suppose that τ is not feasible. Let $t_f \geq 0$ be the earliest time σ reports failure with respect to any legal set of τ-requests. Let R_0 be a legal set of τ-requests such that $(\sigma.R_0)$ reports failure for the first time at time t_f. Consider the finite legal set of τ-requests $R_1 = R_0 - \{(i, t_0) \mid t_0 + d_i > t_f\}$. Recall that $(\sigma.R_1)$ mimics $(\sigma.R_0)$ with respect to all tie breaking choices. Over the interval $[0, t_f)$ $(\sigma.R_0)$ never assigns the processor to a τ-request from $R_0 - R_1$ at a time when a request from R_1 is active. Hence, over $[0, t_f)$, $(\sigma.R_0)$ and $(\sigma.R_1)$ make identical assignments with respect to the τ-requests in R_1. Hence, $(\sigma.R_1)$ too reports failure for the first time at time t_f.

Suppose at time $t_b - 1$, $1 \leq t_b < t_f$, $(\sigma.R_1)$ idles the processor, i.e., $(\sigma.R_1).(t_b - 1) = (0, 0)$. Suppose further that t_b is the largest such integer. If the processor is never idle before t_f, let $t_b = 0$. Let $R_2 = \{(i, t_0) \in R_1 \mid (i, t_0)$ was not assigned the processor by $(\sigma.R_1)$ in $[0, t_b)\}$. Note that R_2 is legal. Note also that $R_2 \cap \{(i, t_0) \mid t_0 < t_b\} = \emptyset$; otherwise, some task would be assigned the processor at time $t_b - 1$. Thus, over $[t_b, t_f)$, $(\sigma.R_1)$ can only assign the processor to τ-requests in R_2. Hence, $(\sigma.R_2)$ will mimic $(\sigma.R_1)$ over $[t_b, t_f)$, and $(\sigma.R_2)$ will report failure for the first time at t_f. Note now that $(\sigma.R_1)$ and thus $(\sigma.R_2)$ never idles the processor over $[t_b, t_f)$.

Now let us review the properties of R_2:

- Each $(i, t_0) \in R_2$ has $t_0 \geq t_b$,
- Each $(i, t_0) \in R_2$ has $t_0 + d_i \leq t_f$,
- $(\sigma.R_2)$ never idles the processor over $[t_b, t_f)$, and
- $(\sigma.R_2)$ reports failure exactly once — at time t_f.

From these properties we immediately obtain that $g_{R_2}(t_b, t_f) > t_f - t_b$. □

Since $g_R(t_1, t_2) > t_2 - t_1$ implies that the processor demand exceeds the time available in $[t_1, t_2)$, we have the following corollary.

Corollary 2.1: Let $\tau = \{T_1, ..., T_n\}$ be a task system. τ is not feasible iff there exist a legal set of τ-requests R and natural numbers $t_1 < t_2$ such that $g_R(t_1, t_2) > t_2 - t_1$.

Corollary 2.1 gives a necessary and sufficient condition for a task system to be infeasible. In the following sections, we will give bounds on t_1 and t_2 for periodic, sporadic, and hybrid task systems. We will also show how to efficiently compute the appropriate $g_R(t_1, t_2)$. Thus, we will have shown upper bounds for the complexities of the feasibility problems for each of the three classes of task systems.

3 Periodic Task Systems

In this section, we show the feasibility problem for periodic task systems to be co-NP-complete in the strong sense. Recall that a periodic task system τ has only one legal set of τ-requests — R_τ. Hence, we first show that given t_1 and t_2, $g_{R_\tau}(t_1, t_2)$ can be computed in linear time. We then show that t_1 and t_2 from Corollary 2.1 can be written down in a polynomial number of bits. It will then follow from Corollary 2.1 that the feasibility problem is in co-NP. From results of Leung and Merrill [LM80], it follows that the problem is co-NP-complete. In order to show the problem to be co-NP-hard in the strong sense, we first show that the Simultaneous Congruences Problem is co-NP-hard in the strong sense. It then follows from the reduction given by Leung and Merrill [LM80] that the feasibility problem for periodic task systems is co-NP-hard in the strong sense; in fact, we are able to show that this result holds even if the processor utilization is bounded above by any fixed positive constant.

Our first lemma allows us to efficiently compute $g_{R_\tau}(t_1, t_2)$, given t_1 and t_2.

Lemma 3.1: Let $\tau = \{T_1, ..., T_n\}$ be a periodic task system. Then

$$g_{R_\tau}(t_1,t_2) = \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}.$$

Proof: Clearly, $g_{R_\tau}(t_1,t_2)$ can be written as $\sum_{i=1}^{n} e_i \cdot c_{R_\tau}(i,t_1,t_2)$, where $c_{R_\tau}(i,t_1,t_2)$ represents the number of requests (i,t_0) in R_τ satisfying $t_0 \geq t_1$ and $t_0 + d_i \leq t$. Thus, $c_{R_\tau}(i,t_1,t_2)$ will be the number of natural numbers k such that k satisfies both

(1) $t_1 \leq s_i + k \cdot p_i$, and

(2) $s_i + k \cdot p_i + d_i \leq t_2$.

(1) yields

$$k \geq \frac{t_1 - s_i}{p_i}.$$

The smallest natural number satisfying this is

$$\max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\}.$$

(2) yields

$$k \leq \frac{t_2 - s_i - d_i}{p_i}.$$

The largest natural number satisfying this is

$$\left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor.$$

$c_{R_\tau}(i,t_1,t_2)$ — the total number of integers k satisfying both (1) and (2) — is therefore:

$$\max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}.$$

We therefore have

$$g_{R_\tau}(t_1,t_2) = \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\}.$$

□

The above lemma shows that $g_{R_\tau}(t_1,t_2)$ can be computed in linear time, given t_1 and t_2. What remains to be shown is that t_1 and t_2 need not be too large. In the next two lemmas we derive an upper bound for t_2 (and hence t_1).

Lemma 3.2: Let $\max_{1 \leq i \leq n}\{s_i\} \leq t_1 < t_2$. Then $g_{R_\tau}(t_1 + P, t_2 + P) = g_{R_\tau}(t_1,t_2)$. (Recall that $P = \mathrm{lcm}\{p_1, ..., p_n\}$.)

Proof: We prove that $c_{R_\tau}(i, t_1 + P, t_2 + P)$ is equal to $c_{R_\tau}(i, t_1, t_2)$ for all i, $1 \leq i \leq n$, where c_{R_τ} is as defined in Lemma 3.1. The lemma will then follow. As in the proof of Lemma 3.1, we have

$$c_{R_\tau}(i, t_1 + P, t_2 + P) = \max\left\{0, \left\lfloor \frac{t_2 + P - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 + P - s_i}{p_i} \right\rceil\right\} + 1\right\}.$$

Since $t_1 \geq s_i$, we may replace $\max\{0, \lceil \frac{t_1+P-s_i}{p_i} \rceil\}$ by $\lceil \frac{t_1+P-s_i}{p_i} \rceil$, and since p_i divides P, we may extract P/p_i from the floor and ceiling terms. Thus,

$$\begin{aligned} c_{R_\tau}(i, t_1 + P, t_2 + P) &= \max\left\{0, \frac{P}{p_i} + \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \frac{P}{p_i} - \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil + 1\right\} \\ &= \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil + 1\right\} \\ &= c_{R_\tau}(i, t_1, t_2). \end{aligned}$$

□

Lemma 3.3: Suppose $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$, $t_1 \geq \max_{1 \leq i \leq n}\{s_i\}$, and $t_2 \geq \max_{1 \leq i \leq n}\{d_i\} + t_1$. Then $g_{R_\tau}(t_1, t_2 + P) > t_2 + P - t_1$ implies $g_{R_\tau}(t_1, t_2) > t_2 - t_1$.

Proof: By Lemma 3.1, we have

$$\begin{aligned} g_{R_\tau}(t_1, t_2 + P) &= \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 + P - s_i - d_i}{p_i} \right\rfloor - \max\left\{0, \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil\right\} + 1\right\} \\ &= \sum_{i=1}^{n} e_i \cdot \max\left\{0, \frac{P}{p_i} + \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil + 1\right\} \end{aligned}$$

as in the proof of Lemma 3.2. Since $P/p_i \geq 0$, we now have

$$\begin{aligned} g_{R_\tau}(t_1, t_2 + P) &\leq \sum_{i=1}^{n} \left(e_i \cdot \frac{P}{p_i} + e_i \cdot \max\left\{0, \left\lfloor \frac{t_2 - s_i - d_i}{p_i} \right\rfloor - \left\lceil \frac{t_1 - s_i}{p_i} \right\rceil + 1\right\}\right) \\ &\leq P\left(\sum_{i=1}^{n} \frac{e_i}{p_i}\right) + g_{R_\tau}(t_1, t_2) \\ &\leq P + g_{R_\tau}(t_1, t_2). \end{aligned}$$

(The last step follows because $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$.) If we now suppose $g_{R_\tau}(t_1, t_2 + P) > t_2 + P - t_1$, we have

$$\begin{aligned} g_{R_\tau}(t_1, t_2) &\geq g_{R_\tau}(t_1, t_2 + P) - P \\ &> t_2 + P - t_1 - P \\ &= t_2 - t_1 \end{aligned}$$

□

As a corollary to the two above lemmas and Corollary 2.1, we have:

Corollary 3.1: Suppose τ is not feasible and $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$. Then there exist t_1 and t_2, $0 \leq t_1 < t_2 \leq 2 \cdot P + \max_{1 \leq i \leq n}\{d_i\} + \max_{1 \leq i \leq n}\{s_i\}$ such that $g_{R_\tau}(t_1, t_2) > t_2 - t_1$.

A nondeterministic algorithm to verify that τ is not feasible now operates as follows. We first determine whether $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$. If so, we guess t_1 and t_2 subject to the constraints of Corollary 3.1. From Corollary 3.1, there exist such t_1 and t_2 such that $g_{R_\tau}(t_1, t_2) > t_2 - t_1$ iff τ is not feasible. From Lemma 3.1, we can evaluate $g_{R_\tau}(t_1, t_2)$ in polynomial time, and accept or reject accordingly. We therefore have the following theorem.

Theorem 3.1: The feasibility problem for periodic task systems is in co-NP.

We will now show that the feasibility problem for periodic task systems is co-NP-hard in the strong sense. In the proof that the feasibility problem is co-NP-hard, Leung and Merrill used a reduction from the complement of the Simultaneous Congruences Problem (SCP). SCP was shown to be NP-complete in the ordinary sense by Leung and Whitehead [LW82]. It was left as an open question whether SCP was NP-complete in the strong sense [LM80, LW82]. This question is important in its own right, because several problems have been shown to be NP-hard in the ordinary sense via reductions from SCP [LW82, Leu89]; furthermore, these reductions are all strong enough to show NP-hardness in the strong sense if SCP is NP-hard in the strong sense. (We discuss these problems further in Section 6.) In what follows, we show SCP to be NP-hard in the strong sense. It then follows from the proof of Leung and Merrill that the feasibility problem is co-NP-hard in the strong sense. Finally, Theorem 3.1 immediately implies that the problem is co-NP-complete.

We will now introduce the Simultaneous Congruences Problem. Let $A = \{(a_1, b_1), ..., (a_n, b_n)\} \subseteq N \times N^+$ and $2 \leq k \leq n$ be given. The *Simultaneous Congruences Problem* is to determine whether there is a subset $A' \subseteq A$ of k pairs and a natural number x such that for every $(a_i, b_i) \in A'$, $x \equiv a_i (\text{mod } b_i)$. In what follows, we show this problem to be NP-complete in the strong sense. Our proof uses the Generalized Chinese Remainder Theorem (see, e.g., [Knu81]), which we now reproduce.

Lemma 3.4: (The Generalized Chinese Remainder Theorem.) Let $A = \{(a_1, b_1), ..., (a_k, b_k)\} \subseteq N \times N^+$. There is an x such that for all $(a_i, b_i) \in A$, $x \equiv a_i (\text{mod } b_i)$ iff for all $1 \leq i < j \leq k$, $a_i \equiv a_j (\text{mod } \gcd(b_i, b_j))$.

Thus, we can conclude that if each pair of distinct pairs (a_i, b_i) and (a_j, b_j) "collides" (i.e., there is an x such that $x \equiv a_i (\text{mod } b_i)$ and $x \equiv a_j (\text{mod } b_j)$), then there is a simultaneous collision of all pairs. This fact is very useful in the construction that follows.

The proof by Leung and Whitehead [LW82] that SCP is NP-hard (in the ordinary sense) consisted of a reduction from the CLIQUE problem [Kar72]. Given an undirected graph $G = (V, E)$ such that $V = \{v_1, ..., v_n\}$, a set of pairs $\{(a_1, b_1), ..., (a_n, b_n)\}$ was constructed such that $a_i \equiv a_j (\text{mod } \gcd(b_i, b_j))$ iff $(v_i, v_j) \in E$. Thus, there is a simultaneous collision of k items iff G has a clique of size k. However, since each a_i and b_i were the product of $O(n^2)$ distinct prime numbers, the values of a_i and b_i were not polynomial in the size of the description of G. In order to overcome this problem, we give an entirely different reduction, from 3-SAT rather than CLIQUE. (See, e.g., [GJ79] for a complete discussion on strong NP-hardness.)

Lemma 3.5: SCP is NP-complete in the strong sense.

Proof: Leung and Whitehead [LW82] have shown SCP to be in NP, so we need only show NP-hardness.

The proof is via a polynomial-time reduction from 3-SAT [Coo71]. Let C be an instance of 3-SAT over variables $x_1, \ldots, x_n$; without loss of generality let $C = \bigwedge_{j=1}^{m} C_j$, $C_j = \bigvee_{k=1}^{3} c_{jk}$, where each c_{jk} is a distinct literal — i.e., either a variable or the negation of a variable. Also without loss of generality, we assume that no clause contains both a variable and its negation.

The intuitive idea of our construction is as follows. We will construct an instance of SCP such that a collision of $m+n$ pairs will give a satisfying assignment for C. For each variable x_i, we will construct a pair (a_{x_i}, b_{x_i}) corresponding to an assignment of *true* to x_i, and a pair $(a_{\bar{x}_i}, b_{\bar{x}_i})$ corresponding to an assignment of *false* to x_i; these two pairs will be constructed so that they do not collide. On the other hand, we will force pairs corresponding to two different variables to collide with each other. Thus, from Lemma 3.4, any simultaneous collision of n of these pairs will correspond to some assignment to all n variables. Now for each clause $c_{j1} \vee c_{j2} \vee c_{j3}$, we will construct three pairs (a_{jk}, b_{jk}), $1 \leq k \leq 3$, one corresponding to each of the three literals in the clause. (a_{jk}, b_{jk}) will be constructed to collide with all other pairs except those pairs corresponding to literals within the same clause and the pair $(a_{\bar{c}_{jk}}, b_{\bar{c}_{jk}})$. Thus, a collision of $m+n$ pairs will occur iff there is an assignment that makes at least one literal in each clause *true*.

More formally, let $p_1, \ldots, p_{m+n}$ denote the first $m+n$ primes greater than 2. From the Prime Number Theorem (see, e.g., [Knu81]), p_{m+n} is bounded by a polynomial in $m+n$. Now for each i, $1 \leq i \leq n$, let

- $a_{x_i} = p_i$;
- $a_{\bar{x}_i} = p_i - 1$; and
- $b_{x_i} = b_{\bar{x}_i} = p_i$.

Note that since each $p_i > 2$, $a_{\bar{x}_i} < a_{x_i} < a_{\bar{x}_{i+1}}$ for $1 \leq i < n$. Now for each j, $1 \leq j \leq m$ and each k, $1 \leq k \leq 3$, let

- $a_{jk} = a_{c_{jk}}$ (defined above); and
- $b_{jk} = b_{c_{jk}} p_{n+j}$.

Clearly, the values of all of the integers defined above are bounded by a polynomial in $m+n$. We will now show that there is a collision of $m+n$ pairs iff C is satisfiable.

$\Rightarrow$: Suppose there is a collision of $m+n$ pairs at some y. For any variable x_i, $\gcd(b_{x_i}, b_{\bar{x}_i}) = p_i$. Since $p_i > 2$, $p_i \not\equiv p_i - 1 (\bmod\ p_i)$. Therefore, from Lemma 3.4, (a_{x_i}, b_{x_i}) and $(a_{\bar{x}_i}, b_{\bar{x}_i})$ do not collide; hence, there is at most one pair corresponding to each variable in the collision at y. For any literal c_{jk}, if $k' \neq k$, $\gcd(b_{jk}, b_{jk'}) = p_{n+j}$ since c_{jk} and $c_{jk'}$ are distinct and noncontradictory. Since $a_{c_{jk}} < p_{n+j}$, $a_{c_{jk'}} < p_{n+j}$, and $a_{c_{jk}} \neq a_{c_{jk'}}$ for all j, k, k', two pairs corresponding to literals in the same clause do no collide. Thus, there is at most one pair corresponding to each clause in the collision at y. Since there are $m+n$ pairs in the collision, there are exactly n pairs corresponding to variables, and m pairs corresponding to clauses. Consider the assignment in which x_i is *true* iff $y \equiv a_{x_i} (\bmod\ b_{x_i})$, for $1 \leq i \leq n$. Let c_{jk} be some literal such that $y \equiv a_{jk} (\mathrm{mod}\ b_{jk})$. (Note that there is exactly one such literal in each clause.) Since $a_{jk} = a_{c_{jk}}$ and b_{jk} is a multiple of $b_{c_{jk}}$, $y \equiv a_{c_{jk}} (\mathrm{mod}\ b_{c_{jk}})$, so c_{jk} is *true* in this assignment. Hence, C is satisfiable.

$\Leftarrow$: Suppose C is satisfiable, and consider some satisfying assignment. Let $A' = \{(a_{x_i}, b_{x_i}) \mid x_i = \mathit{true}\} \cup \{(a_{\bar{x}_i}, b_{\bar{x}_i}) \mid x_i = \mathit{false}\} \cup \{(a_{jk}, b_{jk}) \mid c_{jk} = \mathit{true}$ and for all $k' < k, c_{jk'} = \mathit{false}\}$. It is easily seen by inspection that any two pairs in A' collide. Therefore, from Lemma 3.4, there is a simultaneous collision of the $m + n$ pairs in A'. $\square$

Leung and Merrill [LM80] have given a reduction from SCP to the complement of the feasibility problem that causes only a polynomial increase in the values of the integers involved; it therefore follows from Lemma 3.5 that the feasibility problem is co-NP-hard in the strong sense. For completeness, we now reproduce their reduction.

Theorem 3.2: The feasibility problem for periodic task systems is co-NP-hard in the strong sense.

Proof: Let $\{(a_1, b_1), ..., (a_n, b_n)\}$ and k represent an instance of SCP. Leung and Merrill [LM80] have given the following polynomial-time reduction to the complement of the feasibility problem.[2] For $1 \leq i \leq n$, let

- $s_i = (k-1)a_i$;
- $e_i = 1$;
- $d_i = k - 1$; and
- $p_i = (k-1)b_i$.

Leung and Merrill [LM80] have shown that this task system is feasible iff there is no simultaneous collision of k pairs from the instance of SCP. Furthermore, all of the values in the task system are bounded by a polynomial in the values in the instance of SCP. $\square$

The above theorem implies that there is no pseudo-polynomial-time algorithm for the feasibility problem for periodic task systems unless P = NP. In fact, we can show the following corollary.

Corollary 3.1: The feasibility problem for periodic task systems is co-NP-hard in the strong sense even if the systems are restricted to have processor utilization not greater than ϵ, where ϵ is any fixed positive constant.

Proof: Consider the construction in the proof of Theorem 3.2. If we multiply all of the start times and periods in this construction by the same positive integer, the proof still holds. In particular, if we multiply all of the start times and periods by some positive integer $c \geq n/(\epsilon(k-1)\min\{b_i\})$, then $\sum_{i=1}^{n} e_i/p_i \leq \epsilon$. $\square$

The following theorem now follows immediately from Theorem 3.1 and Corollary 3.1, thus answering another open question posed by Leung and Merrill [LM80].

[2] This reduction is actually slightly different from the one by Leung and Merrill, but the basic idea is the same.

Theorem 3.3: The feasibility problem for periodic task systems is co-NP-complete in the strong sense even if the systems are restricted to have processor utilization not greater than ϵ, where ϵ is any fixed positive constant.

4 Sporadic Task Systems

In this section, we consider the feasibility problem for sporadic task systems. Our strategy is to reduce this problem to the feasibility problem for a special case of periodic task systems known as *synchronous* task systems — periodic task systems in which all start times are zero. Thus, the upper bound given in Theorem 3.1 will hold for sporadic task systems. Although we are unable to show either a co-NP-hard lower bound or a pseudo-polynomial time algorithm for the problem, we are able to give a pseudo-polynomial time algorithm for a very important special case. In particular, let c be any fixed constant, $0 < c < 1$. We show that the feasibility problem for sporadic (or synchronous) task systems having processor utilization no more than c can be solved in $O(n \max\{p_i - d_i\})$ time. The constant of proportionality in this complexity bound increases as c approaces 1; however, for $c = 0.99$, the constant is still reasonably small. Since any system having processor utilization greater than 1 is not feasible, our algorithm runs efficiently for the vast majority of sporadic and synchronous task systems. Note that due to Theorem 3.3, this result does not hold for periodic task systems unless P = NP.

We begin by reducing the feasibility problem for sporadic task systems to the feasibility problem for synchronous task systems. Intuitively, a synchronous task system behaves like a sporadic task system in which each task always makes a request at the earliest legal time. We will be showing that this behavior represents a worst-case behavior of a sporadic task system. To this end, we first show that if a sporadic task system τ is not feasible, any legal set of τ-requests that causes σ to fail at the earliest time also causes σ to keep the processor busy until the first failure is reported. Using this fact, we then show that t_1 from Corollary 2.1 may be chosen to be 0 for sporadic task systems. Given this fact, it is then a straightforward matter to complete the reduction.

Lemma 4.1: Let $\tau = \{T_1, ..., T_n\}$ be a sporadic task system. If τ is not feasible and R is a legal set of τ-requests that causes σ to fail at the earliest time t_f, then $(\sigma.R)$ never idles the processor over $[0, t_f)$.

Proof: Suppose $(\sigma.R)$ does idle the processor in $[0, t_f)$, and let $t_b < t_f$ be the latest such time. Clearly, all τ-requests (i, t_0) such that $t_0 \leq t_b$ are inactive at time t_b. Let $R_1 = \{(i, t_0) \in R \mid t_0 > t_b\}$. Then $(\sigma.R_1)$ is identical to $(\sigma.R)$ over $[t_b + 1, t_f)$, and $(\sigma.R_1)$ reports failure at t_f. Let $R_2 = \{(i, t_0 - t_b - 1) \mid (i, t_0) \in R_1\}$. Then $(\sigma.R_2)$ over $[0, t_f - t_b - 1)$ is identical to $(\sigma.R_1)$ over $[t_b + 1, t_f)$. Thus, $(\sigma.R_2)$ reports failure at $t_f - t_b - 1 < t_f$ — a contradiction. □

Lemma 4.2: Let $\tau = \{T_1, ..., T_n\}$ be a sporadic task system. If τ is not feasible and R is a legal set of τ-requests that causes σ to fail at the earliest time t_f, then $g_R(0, t_f) > t_f$.

Proof: Let $R_1 = \{(i, t_0) \in R \mid t_0 \leq t_f - d_i\}$. Clearly, $g_{R_1}(0, t_f) = g_R(0, t_f)$, and $(\sigma.R_1)$ reports failure at time t_f. From Lemma 4.1, $(\sigma.R_1)$ never idles the processor over $[0, t_f)$; hence, the amount of processor time allocated in $[0, t_f)$ by $(\sigma.R_1)$ is t_f. Since a deadline is missed at t_f, and since no τ-request in R_1 has a deadline later than t_f, $g_{R_1}(0, t_f)$ (and hence $g_R(0, t_f)$) must exceed the amount of processor time allocated by $(\sigma.R_1)$ in $[0, t_f)$; i.e., $g_R(0, t_f) > t_f$. □

The above lemma shows that t_1 from Corollary 2.1 may be chosen to be 0. In the following lemma, we will show that R from Theorem 2.1 may be chosen to be $\mathcal{R} = \cup_{i=1}^{n} \cup_{k \geq 0} \{(i, kp_i)\}$.

Lemma 4.3: Let $\tau = \{T_1, ..., T_n\}$ be a sporadic task system. If τ is not feasible, then for $\mathcal{R} = \cup_{i=1}^{n} \cup_{k \geq 0} \{(i, kp_i)\}$, $g_{\mathcal{R}}(0, t_f) > t_f$ for some $t_f > 0$.

Proof: Clearly, $R = \mathcal{R}$ maximizes $g_R(0, t)$ for all t. The lemma therefore follows from Lemma 4.2. □

Notice that the set $\mathcal{R}$ given in the above lemma is simply $R_{\tau'}$, where τ' is the periodic task system constructed from τ by setting all start times to zero. Thus, Lemma 4.3 shows that the feasibility problem for sporadic task systems reduces to the feasibility problem for synchronous task systems. In particular, we have the following corollary.

Corollary 4.1: Let τ be a sporadic task system, and let τ' be the synchronous task system constructed from τ by setting all start times to zero. Then τ is feasible iff τ' is feasible.

Note that the above reduction is a bijection from the class of sporadic task systems to the class of synchronous task systems; thus, the feasibility problems for the two classes are isomorphic. Furthermore, since synchronous task systems are periodic task systems, the following theorem follows immediately from Theorem 3.1.

Theorem 4.1: The feasibility problem for sporadic task systems is in co-NP.

We do not know at this time whether the feasibility problem for sporadic (or synchronous) task systems is co-NP-hard, or whether there exists a pseudo-polynomial time algorithm for the problem. However, we are able to show that unless P = NP, Corollary 3.1 does not extend to sporadic or synchronous task systems. In particular, we show that for any fixed constant $c < 1$, the feasibility problem for sporadic or synchronous task systems with processor utilization no more than c can be solved in $O(n \max\{p_i - d_i\})$ time. In fact, the constant of proportionality remains fairly small even for $c = 0.99$. Thus, we have an efficient pseudo-polynomial time algorithm for the vast majority of sporadic or synchronous systems.

Theorem 4.2: Let c be a fixed constant, $0 < c < 1$. The feasibility problem for synchronous systems is solvable in $O(n \max\{p_i - d_i\})$ time if $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq c$.

Proof: Let $\tau = \{T_1, ..., T_n\}$ be a synchronous task system such that $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq c$. It follows from Lemma 4.3 that τ is not feasible iff for some $t_f, g_{R_\tau}(0, t_f) > t_f$. We will show that t_f may be chosen to be less than

$$\frac{c}{1-c} \max\{p_i - d_i\}.$$

Since c is a fixed constant, the above value is linear in $\max\{p_i - d_i\}$. Thus, the following algorithm solves the feasibility problem in time $O(n \max\{p_i - d_i\})$:

$t \leftarrow 1$;
$tlim \leftarrow \frac{c}{1-c}\max\{p_i - d_i\}$;
while $t < tlim$ **and** $t \geq \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t-d_i}{p_i} \right\rfloor + 1\right\}$ **do**
    $t \leftarrow t + 1$;
if $t = tlim$ **then**
    return *feasible*
else
    return *not feasible*

Suppose $g_{R_\tau}(0, t_f) > t_f$. Let $I = \{i \mid d_i \leq t_f\}$. From Lemma 3.1, we have

$$\begin{aligned}
t_f &< g_{R_\tau}(0, t_f) \\
&= \sum_{i=1}^{n} e_i \cdot \max\left\{0, \left\lfloor \frac{t_f - d_i}{p_i} \right\rfloor + 1\right\} \\
&= \sum_{i \in I} \left(\left\lfloor \frac{t_f - d_i}{p_i} \right\rfloor + 1 \right) e_i \\
&\leq \sum_{i \in I} \frac{t_f - d_i + p_i}{p_i} e_i \\
&= \sum_{i \in I} \left(\frac{t_f e_i}{p_i} + \frac{(p_i - d_i)e_i}{p_i} \right) \\
&\leq c t_f + c \max\{p_i - d_i\}.
\end{aligned}$$

Solving for t_f, we get

$$t_f < \frac{c}{1-c}\max\{p_i - d_i\}.$$

□

Corollary 4.2: Let c be a fixed constant $0 < c < 1$. The feasibility problem for sporadic task systems is solvable in $O(n \max\{p_i - d_i\})$ time if $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq c$.

5 Hybrid Task Systems

In this section, we consider the feasibility problem for hybrid task systems. Since hybrid task systems are a generalization of periodic task systems, the lower bound of Corollary 3.1 applies. We will provide a matching upper bound to show the problem to be co-NP-complete in the strong sense. Our strategy will be similar to that given in Section 4 in that we will reduce, in some sense, this problem to the feasibility problem for periodic task systems. However, the reduction will not be as strong as the one given in Section 4. We will again show that if a given hybrid task system is not feasible, start times can be given to the sporadic tasks to produce a periodic task system which is not feasible. However, we do not show whether the appropriate start times can be efficiently constructed. Instead, we provide upper bounds that allow them to be guessed efficiently; this procedure is enough to show the feasiblity problem to be in co-NP. The following lemma gives the bounds for the start times.

Lemma 5.1: Let $\tau = \{T_1, ..., T_n, T_{n+1}, ..., T_{n+m}\}$ be a hybrid task system in which tasks $T_1, ..., T_n$ are periodic tasks and $T_{n+1}, ..., T_{n+m}$ are sporadic tasks. If τ is not feasible, then there exist $s_n < p_n, ..., s_{n+m} < p_{n+m}$ such that $\mathcal{R} = \cup_{i=1}^{n+m} \cup_{k \geq 0} \{(i, s_i + kp_i)\}$ causes σ to fail.

Proof: Let R be a legal set of τ-requests that causes σ to fail at the earliest time t_f. From Theorem 2.1, there is a $t_b < t_f$ such that $g_R(t_b, t_f) > t_f - t_b$. For each task T_i, $n \leq i \leq n+m$, let $s_i = t_b \bmod p_i$. Clearly, $s_i < p_i$. Furthermore, it is easily seen that $g_{\mathcal{R}}(t_b, t_f) \geq g_R(t_b, t_f)$ for any legal set of τ-requests R. Thus, $g_{\mathcal{R}}(t_b, t_f) > t_f - t_b$, and $\mathcal{R}$ causes σ to fail. □

A nondeterministic algorithm to determine whether a hybrid task system τ is not feasible is as follows. First, guess start times for the sporadic tasks subject to the constraints of Lemma 5.1, and construct a periodic task system τ' from τ by adding these start times. Now use the nondeterministic algorithm given in Section 3 to determine whether τ' is feasible. Since this algorithm clearly runs in polynomial time, we have the following theorem.

Theorem 5.1: The feasibility problem for hybrid task systems is co-NP-complete in the strong sense even if the systems are restricted to have processor utilization not greater than ϵ, where ϵ is any fixed positive constant.

6 Conclusion

We have shown the feasibility problems for both periodic and hybrid task systems to be co-NP-complete in the strong sense even if their processor utilizations are bounded above by any fixed positive constant. On the other hand, the precise complexity of the feasibility problem for sporadic (or synchronous) task systems remains unknown. The best upper bound known for the problem is co-NP, but the problem has not been shown to be co-NP-hard, even in the ordinary sense. Thus, we still have no evidence to suggest that there is no polynomial-time algorithm for the problem. Furthermore, our strongest lower bounds for periodic and hybrid task systems cannot hold for sporadic task systems unless P = NP; i.e., we have shown that for sporadic task systems whose processor utilization is bounded above by a fixed constant less than one, there is a pseudo-polynomial time feasibility test. This fact might suggest that the feasibility problem for sporadic task systems is computationally easier than for periodic or hybrid task systems.

Our proof that SCP is NP-hard in the strong sense has several ramifications. Leung and Whitehead [LW82] have show that, given a periodic task system and a priority assignment, the problem of deciding whether the schedule produced by that priority assignment is valid is co-NP-hard. They went on to show that the problem of deciding whether there exists a priority assignment that correctly schedules a given periodic task system is also co-NP-hard. More recently, Leung [Leu89] has shown that the problem of deciding if a task system is schedulable on $m \geq 1$ processors by the slack-time (the deadline, or any fixed-priority) algorithm is co-NP-hard for each fixed $m \geq 1$. In all of these cases, the results were shown via a reduction from the complement of SCP. Given our result that SCP is NP-hard in the strong sense, each of these results may now be strengthened to co-NP-hard in the strong sense. For a more complete discussion of these problems, see [LW82, Leu89].

References

[BHR90] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[BMR90] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, 1990.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. of the 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[Der74] M. Dertouzos. Control robotics: the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[HL88] K. Hong and J. Leung. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium*, pages 244–250, 1988.

[JSM91] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pages 129–139, 1991.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Knu81] D. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison Wesley, second edition, 1981.

[Lab74] J. Labetoulle. Some theorems on real time scheduling. In E. Gelenbe and R. Mahl, editors, *Computer Architecture and Networks*, pages 285–293. North-Holland, 1974.

[Leu89] J. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4:209–219, 1989.

[LL73] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *JACM*, 20:46–61, 1973.

[LM80] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[LM81] E. Lawler and C. Martel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters*, 12:9–12, 1981.

[LSS87] J. Lehoczky, L. Sha, and J. Stronider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the Real-Time Systems Symposium*, pages 261–270, 1987.

[LW82] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[Mok83] A. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, MIT Laboratory for Computer Science, 1983.

[SLS88] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of the Real-Time Systems Symposium*, pages 251–258, 1988.

[SSL89] B. Sprunt, L. Sha, and J. Lehoczky. Scheduling sporadic and aperiodic events in a hard real-time system. Technical Report ESD-TR-89-19, Carnegie Mellon University, 1989.