Algorithms and Complexity for Periodic Real-Time Scheduling

Vincenzo Bonifaci[*][†] Ho-Leung Chan[‡] Alberto Marchetti-Spaccamela[§] Nicole Megow[*]

Abstract

We investigate the preemptive scheduling of periodic tasks with hard deadlines. We show that, even in the uniprocessor case, no polynomial time algorithm can test the feasibility of a task system within a constant speedup bound, unless $\mathsf{P} = \mathsf{NP}$. This result contrasts with recent results for sporadic task systems. For two special cases, synchronous task systems and systems with a constant number of different task types, we provide the first polynomial time constant-speedup feasibility tests for multiprocessor platforms. Furthermore, we show that the problem of testing feasibility is coNP-hard for synchronous multiprocessor task systems. The complexity of some of these problems has been open for a long time.

We also propose a profit maximization variant of the feasibility problem, where every task has a non-negative profit, and the goal is to find a subset of tasks that can be scheduled feasibly with maximum profit. We give the first constant-speed, constant-approximation algorithm for the case of synchronous task systems, together with related hardness results.

1 Introduction

We consider problems concerned with the feasibility of scheduling a set of periodic tasks in a hard real-time environment. A real-time task system consists of a finite number of tasks, each of which generates an infinite sequence of recurring jobs. There is given one or multiple processors, each of which can process only one job at the time. Now, each job must be executed by the system, possibly with preemptions and migration, before its deadline.

In a *periodic task system* I, a task $i \in I$ is defined by a quadruple (r_i, c_i, d_i, p_i), where the offset (or starting time) r_i specifies the time instant at which the first job of task i is released, the execution time (or size) c_i defines the processing requirement for each job of task i, the relative deadline d_i represents the time interval between the release of a job and its hard deadline, and the period p_i specifies the temporal separation between the release of two successive jobs of task i. Thus, the k-th job of task i is released at time $r_i + (k-1)p_i$ and has to be processed for c_i time units before time $r_i + (k-1)p_i + d_i$.

In this paper, we restrict our attention to *constrained-deadline* periodic task systems, in which the quite common assumption is made that $d_i \le p_i$, for all $i \in I$. Moreover, we assume all input parameters have integer value. In that case, we can restrict ourselves without loss of generality to discrete schedules [5], where job preemptions can occur only at integral times. Preemptions and migrations are allowed at no cost.

A task system is said to be *feasible* if there exists a schedule such that each job completes its execution requirement before its deadline. The system is called A-*schedulable* if algorithm A constructs a feasible schedule for the task system. The *feasibility problem* is concerned with deciding if a given task system is feasible.

A well-known necessary condition for feasibility of a uniprocessor task system I is that $U(I) := \sum_{i \in I} c_i/p_i \le 1$. Quantity $U(I)$ is called the *utilization* of the task system and c_i/p_i is called the utilization of task i. Unfortunately, the condition $U(I) \le 1$ is far from being sufficient for feasibility. In fact, it is known that deciding feasibility for a periodic task system on one processor is strongly coNP-hard even when the utilization is bounded above by any constant less than 1 [5, 18].

In the hope of overcoming hardness results, it is meaningful to relax the accuracy requirements of the feasibility problem slightly. Therefore, the concept of approximate feasibility has been introduced [7], which can be interpreted as a form of *resource augmentation* [13, 24]. For a fixed speedup parameter $\sigma \ge 1$, the problem of deciding σ-approximate feasibility is as follows.

[*]`{bonifaci,nmegow}@mpi-inf.mpg.de`. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[†]Università dell'Aquila, Italy.

[‡]`hlchan@cs.hku.hk`. The University of Hong Kong, Hong Kong.

[§]`alberto@dis.uniroma1.it`. Sapienza Università di Roma, Italy. The work of A. Marchetti-Spaccamela was partially supported by the ICT Programme of the European Union under contract ICT-2008-215270 (FRONTS).

σ-Approximate Feasibility
Input: a periodic task system I and a positive integer m.
Output: an answer YES or NO such that

- YES implies that system I is feasible on m speed-σ processors, and
- NO implies that I is not feasible on m speed-1 processors.

We also consider the following natural optimization variant of the feasibility problem, in which we ask for a maximum weight subset of tasks that can be scheduled feasibly.

Maximum Weight Feasible Subsystem (MaxFS)
Input: a periodic task system I, a positive integer m, weights $w : I \to \mathbb{Q}_+$.
Output: subset of tasks $S \subseteq I$ such that S is feasible on m speed-1 processors.
Objective: maximize $\sum_{i \in S} w_i$.

Clearly, MaxFS is not easier than the feasibility problem from the point of view of exact solutions. However, an approximate solution to the weight maximization problem does not immediately yield a useful answer to the feasibility problem, so the optimization problem might be easier from the point of view of approximations.

As in the case of the feasibility problem, we analyze MaxFS using resource augmentation. An algorithm A is a *σ-speed ρ-approximation* algorithm for MaxFS if, on any input, A returns a subset of tasks that is feasible on m speed-σ processors and has total weight at least $1/\rho$ times the weight of any subset of tasks that is feasible on m speed-1 processors.

Previous work. For periodic task systems, most of the existing results on feasibility testing concern the uniprocessor case. In the uniprocessor setting, the well-known Earliest Deadline First (EDF) algorithm, that schedules jobs in order of their absolute deadline, is optimal in the sense that any feasible system is EDF-schedulable. In spite of that, the feasibility problem is strongly coNP-hard: the reason here is that the first failure of EDF might occur after an exponential amount of time [5, 18].

In the special case of uniprocessor scheduling with a constant number of task types, Baruah et al. [5] show how to solve the feasibility problem in polynomial time, by formulating it as an integer linear program of constant dimension.

Another interesting special case is that of *synchronous task systems.* In this case all tasks start generating jobs simultaneously, that is, $r_i = 0$ for all $i \in I$. In this setting, Albers and Slomka [1] provide a $(1+\epsilon)$-approximate feasibility test for synchronous task systems on a single processor, for any $\epsilon > 0$. A pseudopolynomial time feasibility test is possible when $U(I) \leq \mu$ for some constant $\mu < 1$ [5]. The complexity of the exact – that is, 1-approximate – feasibility problem for synchronous task systems has been open for a long time [5].

In the multiprocessor case, the feasibility problem seems even harder. The best algorithm known uses exponential time *and* space [16]. Phillips et al. [24] proved that EDF, when run on m processors of speed $2 - 1/m$, can meet all deadlines of a system that is feasible on m speed-1 processors; but, as before, this does not yield an efficient test for feasibility or approximate feasibility. However, recently some approximate feasibility tests have been derived for sporadic task systems [4, 6]. Sporadic tasks are defined similarly to periodic tasks, except that no offsets are given and the period defines the minimum (as opposed to exact) temporal separation between the release of two successive jobs of one task. Consequently, a sporadic task system implicitly defines an infinite set of job sequences, and the system is called feasible when *all* the job sequences compatible with its parameters are schedulable.

The weight maximization problem is very natural and relevant in various applications, which is also reflected by the attention that related scheduling problems received in the past, see e.g. [3, 11, 14, 15] and references therein. The crucial difference between previous considerations and our setting lies in the recurrence of real-time tasks. We are not aware of any weight maximization results for periodic task systems.

Our contribution. We show that σ-Approximate Feasibility is coNP-hard for periodic task systems for any $\sigma \leq n^{1-\epsilon}$, where n is the number of tasks and ϵ is any positive real number, even on a single processor. Assuming P$\neq$NP, this rules out any polynomial time algorithm for testing feasibility with a constant speedup factor. This result is in strong contrast to previous approximability results for sporadic task systems [1, 4, 6].

To solve the complexity status of σ-Approximate Feasibility, we reduce from a maximization variant of the number theoretic Simultaneous Congruences problem. This problem is interesting by itself and we are not aware of any hardness of approximation result for it. We prove that this problem is NP-hard to approximate within a factor $n^{1-\epsilon}$, for any $\epsilon > 0$, where n is the number of congruences.

In the special case of synchronous systems we show that 1-Approximate Feasibility for synchronous multiprocessors task systems is coNP-hard. To this aim we first define and study Least Common Multiple Packing, a number theoretic problem that given a set I of integers and two integers k and L requires to find a set S, $S \subseteq I$ and $|S| > k$, such that the least common multiple of integers in S is less than L. Independently of this work, Eisenbrand and Rothvoss proved (in these same proceedings) that even the uniprocessor case of the problem is coNP-hard [9].

We complement our negative results for periodic real-time systems with the first constant approximation algorithms for two restricted models. We provide a polynomial time $(2-1/m)$-approximate test for periodic multiprocessor task systems with a constant number of different task types. Similar to the uniprocessor test by [5], we decide feasibility by solving integer linear programs of constant dimension. We utilize a necessary condition based on an estimate of the total workload in any interval, that was introduced recently for sporadic task systems [6]. For synchronous periodic multiprocessor task systems, we give a $(2-1/m+\epsilon)$-approximate feasibility test that runs in time polynomial in the input and $1/\epsilon$. To obtain the positive result, we introduce a stronger version of the above mentioned estimate of the total workload per interval.

We already mentioned that MaxFS is not easier than the problem of deciding the feasibility of a task system. We show that it is NP-hard to approximate MaxFS to within $n^{1-\epsilon}$, even in the case of a uniprocessor and unit task weights. Moreover, we show that MaxFS is NP-hard even in the strongly restricted setting of synchronous arrivals with implicit deadlines (i.e., $d_i = p_i$) and $w_i = c_i/p_i$.

On the positive side, we give the first constant-speed, constant-approximation algorithms. For synchronous uniprocessor task systems, we give a 2-speed $(4+\epsilon)$-approximate algorithm with running time polynomial in the input and $1/\epsilon$. For the special case where weight equals utilization, we give an improved algorithm that is 4-speed 1-approximate.

Our results for the approximate feasibility problem and the optimization problem are summarized in Tables 1 and 2, respectively.

2 The approximate feasibility problem

2.1 Arbitrary periodic task systems. In this section we prove hardness of approximation for the feasibility problem for periodic task systems. In earlier complexity investigations showing that the problem is coNP-hard, Leung and Merrill [18] reduce from the Simultaneous Congruences Problem. This problem is known to be NP-complete, even in the strong sense [5, 19]. We consider the following natural maximization variant of this decision problem.

Maximum Simultaneous Congruences (MaxSC)
Input: $a_1, \dots, a_n \in \mathbb{N}$, $b_1, \dots, b_n \in \mathbb{N}$.
Output: $S \subseteq \{1, \dots, n\}$ such that the set $\{t \in \mathbb{N} : t \equiv a_i \pmod{b_i}$ for all $i \in S\}$ is nonempty.
Objective: Maximize $|S|$.

This problem can be seen as a Maximum Feasible Subsystem type of problem [10], with univariate congruences in place of multivariate linear equalities. We show the following inapproximability result for MaxSC.

Theorem 2.1. *For all $\epsilon > 0$, it is NP-hard to approximate MaxSC within a factor $n^{1-\epsilon}$.*

Proof. We give an approximation preserving reduction from Maximum Independent Set, which is known to be NP-hard to approximate within $n^{1-\epsilon}$ [26]. Consider a graph $G(V, E)$ and let $V = \{1, 2, \dots, n\}$. We set $a_i = i$ for $i \in V$. Moreover, to every edge $e \in E$ we associate a distinct prime number $\pi(e) > n$. This can be done in polynomial time, for example it is known [23] that there are at least n^2 prime numbers in the range $(n, 4n^4]$. We finally define $b_i := \prod_{e \in \delta(i)} \pi(e)$.

Now if $(i, j) \notin E$ then $\gcd(b_i, b_j) = 1$ and then $a_i \equiv a_j \pmod{\gcd(b_i, b_j)}$. If $(i, j) \in E$ then $\gcd(b_i, b_j) = \pi((i, j)) > \max(a_i, a_j)$ so that $a_i \not\equiv a_j \pmod{\gcd(b_i, b_j)}$. Thus, by the Generalized Chinese Remainder Theorem, see e.g. [2], a set S of congruences is satisfiable if and only if S is an independent set in G. The theorem follows. $\square$

Theorem 2.2. *For any $\epsilon > 0$ and $1 \le \sigma \le n^{1-\epsilon}$, σ-Approximate Feasibility is coNP-hard, even in the single processor case.*

Proof. We show that a polynomial time algorithm for σ-Approximate Feasibility could be used to distinguish between systems that admit k simultaneously satisfiable congruences, and systems for which no set of k/σ simultaneously satisfiable congruences exists, which is NP-hard by Theorem 2.1.

We associate a task to every congruence. For each $1 \le i \le n$, we set $r_i = k \cdot a_i$, $c_i = \sigma$, $d_i = k$, $p_i = k \cdot b_i$. We also add an extra task with $r_{n+1} = 0$, $c_{n+1} = 1$, and $d_{n+1} = p_{n+1} = k$. Without loss of generality we assume that σ is an integer (otherwise we round it up).

If k congruences are simultaneously satisfiable, then there is a time t when k jobs are released simultaneously, meaning that during the interval $[t, t+k]$ at least $\sigma \cdot k + 1 > \sigma \cdot k$ units of work would have to be processed, and thus, the task system is infeasible for a speed-σ machine. Hence, the algorithm must output NO.

On the other hand, if there is no set of k/σ simultaneously satisfiable congruences, then in every interval $[t, t+k]$, the total work to be processed is an integer strictly less than $\sigma \cdot (k/\sigma) + 1$, meaning that it

	Uniprocessor			Multiprocessor		
	σ	Complexity		σ	Complexity	
Arbitrary systems	$n^{1-\epsilon}$	coNP-hard	$*$	$n^{1-\epsilon}$	coNP-hard	$*$
Synchronous systems	$1+\epsilon$	P	[1]	$2-1/m+\epsilon$	P	$*$
				1	coNP-hard	$*$
Constant n. of task types	1	P	[5]	$2-1/m$	P	$*$

Table 1: Results for σ-Approximate Feasibility. Results that are given in this paper are marked with $*$. Here n is the number of tasks and ϵ is any positive real constant.

is at most k and so it can be processed by a unit speed machine using (for example) EDF. Thus the algorithm must output YES. $\square$

2.2 Approximate feasibility tests for a constant number of task types. We have seen that Theorem 2.2 defeats the hope for any constant-approximate polynomial time algorithm for deciding the feasibility of an arbitrary periodic task system. However, for the special case in which the system consists of a constant number of different task types, we derive a polynomial time feasibility test that decides either that EDF provides a feasible schedule on m processors of speed $2-1/m$, or, that the system is infeasible on m speed-1 processors. In this model, tasks belonging to the same task type have identical parameters (offset, execution time, relative deadline and period).

In the context of sporadic task systems, Bonifaci et al. [6] introduced a lower bound on the total processing requirement of a task system in an interval, which they called *forward forced demand* (ffd).

Definition 2.1. (Forward Forced Demand) *Let I be a task system and $\Delta = [t_1, t_2]$ be an interval. Let k_i be the number of jobs of task i that are released strictly before t_1, and let k'_i be the number of jobs of task i that are released and due within the interval Δ. Then*

$$\mathrm{ffd}_I(\Delta) = \sum_{i\in I} k'_i c_i + (c_i - (t_1 - r_i - (k_i-1)p_i)^+)^+.$$

The following results show that the forward forced demand approximately characterizes the EDF-schedulability on multiple processors of a particular speed. They were originally formulated for sporadic task systems, but can be easily seen to apply to periodic task systems as well.

Proposition 2.1. ([6]) *If a periodic task system I is feasible on m unit speed processors, then $\mathrm{ffd}_I(\Delta) \le m\,\|\Delta\|$ for any interval Δ.*

Theorem 2.3. ([6]) *Let I be a periodic task system, $\sigma \ge 1$, and $m \in \mathbb{N}$. If I is not EDF-schedulable on m speed-σ processors, then there is an interval Δ such that $\mathrm{ffd}_I(\Delta)/\|\Delta\| > m(\sigma-1)+1$.*

With these prerequisites we can state our result.

Theorem 2.4. *For periodic task systems with a fixed number of distinct types of tasks on m processors, there is a polynomial time algorithm solving σ-Approximate Feasibility, for any $\sigma \ge 2-1/m$.*

Proof. Given a periodic task system I, let $\bar{n}$ denote the number of distinct types of tasks, and let n_i, for $i = 1,\dots,\bar{n}$, denote the number of tasks of the i-th task type. We use $\mathrm{lcm}(p_1,\dots,p_{\bar{n}})$ to denote the least common multiple of the periods. Assume there is an interval $\Delta := [t_1, t_2]$ such that $\mathrm{ffd}(\Delta) > m\|\Delta\|$. Without loss of generality we can assume that $r_i \le t_1$ for each task $i \in I$; if not, we can increase both t_1 and t_2 by some multiple of $\mathrm{lcm}(p_1,\dots,p_{\bar{n}})$ and the ffd does not decrease.

We construct a system of (linear and non-linear) inequalities that characterizes such an interval Δ. By Proposition 2.1, a feasible solution of this system implies that I is infeasible.

$$r_i + k_i p_i \ge t_1, \tag{2.1}$$

$$r_i + (k_i-1)p_i < t_1, \tag{2.2}$$

$$r_i + k_i p_i + (k'_i - 1)p_i + d_i \le t_2, \tag{2.3}$$

$$r_i \le t_1, \tag{2.4}$$

$$\sum_{i\in I} n_i k'_i c_i + n_i(c_i - (t_1 - r_i - (k_i-1)p_i)^+)^+ > m(t_2 - t_1) \tag{2.5}$$

$$t_1, t_2, k_i, k'_i \in \mathbb{Z}^+ \tag{2.6}$$

The variables of this system of inequalities are t_1, t_2, the end points of the interval Δ, and k_i and k'_i, for $i = 1,\dots,\bar{n}$. Here, k_i is the number of jobs of

	Uniprocessor			
	Speed	Approximation	Complexity	
Arbitrary systems	1	$n^{1-\epsilon}$	NP-hard	$*$
	$n^{1-\epsilon}$	1	coNP-hard	$*$
Synchronous systems	2	$4+\epsilon$	P	$*$
Synchronous syst., $w_i = c_i/p_i$	4	1	P	$*$
	1	1	NP-hard	$*$

Table 2: Results for Maximum Weight Feasible Subsystem. Results that are given in this paper are marked with $*$. Here n is the number of tasks and ϵ is any positive real constant.

task i that are released strictly before t_1, which is ensured by (2.1) and (2.2). Variable k_i' is the number of jobs of task i that are released and due within the interval $[t_1, t_2]$, see (2.3). The left hand side of Inequality (2.5) expresses ffd(Δ) (cf. Definition 2.1), and thus, (2.5) enforces that the workload inequality in Proposition 2.1 is violated, i.e., $\text{ffd}_I(\Delta) > m\|\Delta\|$.

The expression of $\text{ffd}_I(\Delta)$ on the left hand side of inequality (2.5) contains the non-linear term $g_i := (c_i - (t_1 - r_i - (k_i - 1)p_i)^+)^+$. Notice that by constraint (2.2) g_i can take only one of two values for any $i \in I$:

$$g_i = \begin{cases} c_i - (t_1 - r_i - p_i(k_i-1)) & \\ \qquad \text{if } c_i - (t_1 - r_i - p_i(k_i-1)) > 0 & (2.7') \\ 0 & \\ \qquad \text{if } c_i - (t_1 - r_i - p_i(k_i-1)) \le 0 & (2.7''). \end{cases}$$

The idea now is to guess, for each i, which of the two cases occurs. That is, we consider $2^{\bar{n}}$ integer linear programs. Every such program consists of the inequalities (2.1)–(2.6) above, with inequality (2.5) being simplified in the appropriate way, plus inequality (2.7') or (2.7") for each i, depending on the guess for the corresponding term g_i.

For any choice of g_i's, for $i = 1, \ldots, \bar{n}$, this yields a system of $5\bar{n}+1$ linear inequalities. Since $\bar{n}$ is fixed, we obtain integer linear programs with a constant number of variables and inequalities. Therefore, for each of these programs, we can verify in polynomial time if there is an integral solution; see Lenstra [17].

If any of these integer programs has a feasible solution, then we have found an overloaded interval Δ which proves that the task system is infeasible by Proposition 2.1. Otherwise, such an interval cannot exist and thus Theorem 2.3 implies that EDF yields a feasible schedule on m processors of speed $2 - 1/m$. $\square$

2.3 Approximate feasibility tests for synchronous task systems. In the special case of synchronous task systems, where all tasks have equal starting times, we show coNP-hardness and give a constant approximate feasibility test.

To derive hardness, we reduce from the following number theoretic problem. We believe that this problem is of independent interest.

Least Common Multiple Packing
Input: a sequence $q_1, \ldots, q_m$ of positive integers and two positive integers k and L.
Question: is there $S \subseteq \{1, 2, \ldots, m\}$ such that $|S| > k$ and $\text{lcm}\{q_i : i \in S\} \le L$?

Theorem 2.5. Least Common Multiple Packing *is* NP*-hard.*

Proof. A (k,n)-*Mignotte sequence* [21] is a set of n pairwise coprime integers $\pi_1 < \pi_2 < \ldots < \pi_n$ such that the product of any k of them is larger than the product of any $k-1$ of them, that is $\Pi_{1 \le i \le k}\pi_i > \Pi_{1 \le i \le k-1}\pi_{n-i+1}$. Such a sequence can be constructed in expected time that is polynomial in n, by the well-known method of sampling random primes in the interval $(2^n, 2^{n+1})$. In fact, it can also be constructed in deterministic polynomial time (see Lemma A.2 in the Appendix).

We reduce from the decision version of Maximum Clique to Least Common Multiple Packing. Given a graph $G = (\{1, 2, \ldots, n\}, E)$ and an integer s, we construct an $(s+1, n)$-Mignotte sequence $\pi_1 < \ldots < \pi_n$ and define $m = |E|$ integers by setting $q_e := \pi_i \cdot \pi_j$ for each $e = (i,j) \in E$. We also set $L := \Pi_{1 \le i \le s}\pi_{n-i+1}$ and $k := \binom{s}{2} - 1$.

Now if G has a s-clique, and S is the corresponding set of $k+1$ edges, since S spans exactly s vertices we have $\text{lcm}\{q_i : i \in S\} \le \Pi_{1 \le i \le s}\pi_{n-i+1} = L$. Conversely, if G has no s-clique, any set S of at least $k+1$ edges must span at least $s+1$ vertices, so that $\text{lcm}\{q_i : i \in S\} \ge \Pi_{1 \le i \le s+1}\pi_i > L$. $\square$

We can now proceed to prove hardness of the feasibility problem for synchronous systems.

Theorem 2.6. 1-Approximate Feasibility *is* coNP*-hard for synchronous multiprocessor task systems.*

Proof. We reduce from Least Common Multiple Packing. Given $q_1, \dots, q_m, k, L \in \mathbb{N}$ we create a system of $m + k$ tasks. For $1 \le i \le m$, task i has the following parameters: $r_i = 0$, $c_i = q_i - 1$, $d_i = q_i - 1$, $p_i = q_i$. Notice that each job from any of these tasks must be started as soon as it is released in order to meet its deadline. Thus, m processors are certainly necessary for feasibility. We will define the remaining k tasks in such a way that it will be possible to fit them in the unused time slots on the m processors if and only if there is no solution to the Least Common Multiple Packing instance.

For any $t \ge 0$ and $1 \le i \le m$, let

$$h_i(t) := \begin{cases} 1 & \text{if } t \equiv -1 \pmod{q_i} \\ 0 & \text{otherwise.} \end{cases}$$

That is, $h_i(t) = 1$ if and only if task i does *not* have to be scheduled during interval $[t, t+1]$. Also let $h(t) := \sum_{1 \le i \le m} h_i(t)$; this is the total number of "free" processor slots during $[t, t+1]$. We now define the remaining k identical tasks by setting, for each $j = m+1, \dots, m+k$: $r_j = 0$, $c_j = (1/k) \cdot \sum_{0 \le t < L} h(t)$, $d_j = L$, $p_j = \operatorname{lcm}\{q_1, \dots, q_m\}$. We remark that all these parameters can be computed in polynomial time, in particular $c_j = (1/k) \sum_{1 \le i \le m} \lfloor L/p_i \rfloor$.

For the analysis, consider the quantity $H := \max_{0 \le t < L} h(t)$. This is the maximum number of slots that are simultaneously free at any time between 0 and L. Now, the total amount of work needed for the additional k tasks is $\sum_{0 \le t < L} h(t)$. However, because there are only k additional tasks and we cannot process a task simultaneously on more than one processor, the total available time is in fact $\sum_{0 \le t < L} \min(h(t), k)$. So it will be possible to schedule all the tasks if and only if $H \le k$.

For a set $S \subseteq \{1, \dots, m\}$, the minimum t for which $h_i(t) = 1$ for all $i \in S$ is easily seen to be $\operatorname{lcm}\{q_i : i \in S\} - 1$. Thus, $H \le k$ if and only if there is no set S such that $|S| > k$ and $\operatorname{lcm}\{q_i : i \in S\} - 1 < L$, that is, if and only if the instance of Least Common Multiple Packing has no solution. □

In the remainder of this section, we give an approximate feasibility test for synchronous systems. To this aim, we introduce a strengthened formulation of the forward forced demand (recall Definition 2.1). The definition of ffd for any interval $[t_1, t_2]$ only considers the demand of jobs which have their deadline in $[t_1, t_2]$. This may neglect the demand of some job i_k with deadline in $(t_2, t_2 + c_i)$ that necessarily must be scheduled also within $[t_1, t_2]$.

Definition 2.2. (Extended ffd) *Consider a task system I where a task $i \in I$ consists of jobs i_k, $k = 0, 1, \dots,$ with corresponding release dates $r(i_k) := r_i + kp_i$ and deadlines $d(i_k) := r_i + kp_i + d_i$. Given an interval $\Delta := [t_1, t_2]$ we define*

$$\operatorname{effd}_I(i_k, \Delta) := (c_i - (t_1 - r(i_k))^+ - (d(i_k) - t_2)^+)^+,$$

$$\operatorname{effd}_I(i, \Delta) := \sum_k \operatorname{effd}_I(i_k, \Delta),$$

$$\operatorname{effd}_I(\Delta) := \sum_{i \in I} \operatorname{effd}_I(i, \Delta).$$

It is easy to see that the extended forward forced demand of an interval is a lower bound on the total processing requirement of a feasible task system in that interval.

Proposition 2.2. *If a periodic task system I is feasible on m unit speed processors, then* $\operatorname{effd}_I(\Delta) \le m \|\Delta\|$ *for any interval Δ.*

The following lemma shows that $\operatorname{effd}(\Delta)/\|\Delta\|$ is maximized for an interval Δ starting at time 0.

Lemma 2.1. *For any synchronous periodic task system I,*

$$\max_\Delta \frac{\operatorname{effd}_I(\Delta)}{\|\Delta\|} = \max_t \frac{\operatorname{effd}_I([0, t])}{t}.$$

Proof. Let $\Delta := [t_1, t_2]$ be such that $\operatorname{effd}_I(\Delta)/\|\Delta\|$ is maximized. Clearly, $\operatorname{effd}_I(\Delta)/\|\Delta\| \ge \operatorname{effd}_I([0, t])/t$. We construct an instance I' which differs from I only in the start times: for each task i let $0 \le \delta_i \le p_i$ denote the value by which we must increase r_i such that a job is released at t_1. Let $r_i' = r_i + \delta_i$. We show that $\operatorname{effd}_{I'}(\Delta) \ge \operatorname{effd}_I(\Delta)$ on a task by task basis.

To that end, consider some task i and δ_i. The crucial observation is that the change in the effd value when increasing start times is due to (i) the decreased contribution of the last job i_ℓ released strictly before t_2 and (ii) the increased contribution of the last job i_k released strictly before t_1. No other jobs contribution is affected. We observe that (i) the decrease in the contribution of i_ℓ is bounded by $\min\{\delta, c_i\}$, and (ii), the increased contribution of i_k is at least $\min\{\delta, c_i\}$. We omit details.

Thus, $\operatorname{effd}_{I'}(\Delta) \ge \operatorname{effd}_I(\Delta)$. Since we consider periodic task systems, this implies that the expression $\operatorname{effd}_I(\Delta)/\|\Delta\|$ is maximized on any interval of length $\|\Delta\|$ if all tasks simultaneously release a job at the beginning of the interval. By definition, in synchronous systems such an interval is $[0, \|\Delta\|]$. Thus, $\mathrm{effd}_{I'}(\Delta)$ corresponds to the $\mathrm{effd}_I([0, \|\Delta\|])$, which implies the lemma. $\square$

With the previous lemma we can simplify the expression for the maximum extended forward forced demand per time interval as follows.

PROPOSITION 2.3. *Given a synchronous periodic task system I and $t \in \mathbb{N}$, then*

$$\max_{\Delta:\|\Delta\|=t} \mathrm{effd}_I(\Delta) = \sum_{i\in I} k_i c_i + (c_i - (k_i p_i + d_i - t)^+)^+ ,$$
$$\text{where } k_i := \left\lfloor \frac{t + p_i - d_i}{p_i} \right\rfloor .$$

THEOREM 2.7. *Let $\epsilon > 0$. Given a synchronous periodic task system for m processors, there is an algorithm solving σ-APPROXIMATE FEASIBILITY, for any $\sigma \geq 2 - 1/m + \epsilon$, with running time that is polynomial in the input size and $1/\epsilon$.*

Proof. Our goal is to approximate the so-called maximum total load in any time interval, that is, the quantity $\lambda^* := \max_t \mathrm{effd}_I([0, t])/t$. By Proposition 2.3, this value in a synchronous periodic task system equals the value of the maximum total load in a sporadic task system, see [6, Lemma 4.1]. Bonifaci et al. [6] provide a fully polynomial time scheme that, for any $\epsilon > 0$, computes λ such that $\lambda^*/(1 + \epsilon) \leq \lambda \leq \lambda^*$. Now we compare λ with m: if $\lambda > m$, then there must be an interval Δ such that $\mathrm{effd}(\Delta) > m \|\Delta\|$, and then by Proposition 2.2 the task system cannot be feasible on m unit speed machines. If $\lambda \leq m$, then for any interval Δ, $\mathrm{ffd}(\Delta) \leq \mathrm{effd}(\Delta) \leq (1 + \epsilon)m \|\Delta\|$, and by Theorem 2.3 (with $\sigma = 2 - 1/m + \epsilon$) the task system must be EDF-schedulable on m speed-$(2 - 1/m + \epsilon)$ machines. $\square$

3 The maximum weight feasible subsystem problem

3.1 Hardness.
The lower bounds of this section carry over directly to the multiprocessor case.

THEOREM 3.1. *For any $\epsilon > 0$, it is NP-hard to approximate MAXFS within a factor of $n^{1-\epsilon}$ in the single processor case.*

Proof. We give an approximation preserving reduction from MAXIMUM CLIQUE, which is NP-hard to approximate within $n^{1-\epsilon}$, where n is the number of vertices in the graph [12]. Using the same construction as in Theorem 2.1, we obtain numbers a_i, b_i such that:

- if $(i, j) \in E$ then $a_i \not\equiv a_j \pmod{\gcd(b_i, b_j)}$;
- if $(i, j) \notin E$ then $a_i \equiv a_j \pmod{\gcd(b_i, b_j)}$.

We now associate a task to every node i. We set, for all $1 \leq i \leq n$, $r_i = a_i$, $c_i = 1$, $d_i = 1$, $p_i = b_i$. Now any feasible subset of tasks must be a clique in the original graph (otherwise there would be a time where at least two jobs are released simultaneously and thus cannot be completed in time by a single unit-speed processor), and vice versa any clique in the original graph determines a subset of tasks that is feasible, because no two tasks are ever released at the same time and all completion times are one. $\square$

THEOREM 3.2. *For synchronous systems, MAXFS is NP-hard in the single processor case. This holds even when $w_i = c_i/p_i$ for all task $i \in I$.*

Proof. We reduce from SUBSET SUM: given integers $a_1, \ldots, a_n$ and a target integer A, decide if there is a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i\in S} a_i = A$. We set $c_i = a_i$, $d_i = p_i = A$, $r_i = 0$ for all i. In a periodic task system where $d_i = p_i$ for all i, a subset S of tasks is feasible on one processor if and only if $\sum_{i\in S} c_i/p_i \leq 1$, that is, $\sum_{i\in S} a_i \leq A$ [20]. Now an optimal subset of tasks has total weight 1 if and only if there is a subset S such that $\sum_{i\in S} a_i = A$. $\square$

3.2 Approximation algorithms for synchronous systems.
The above lower bounds motivate us to focus on synchronous systems, for which we present two algorithms. For general weights, we give a 2-speed $(4 + \epsilon)$-approximate algorithm. For the special case where $w_i = c_i/p_i$ for all tasks i, we give an improved 4-speed 1-approximate algorithm. We first state an observation about the total size of jobs generated by a task up to certain time.

OBSERVATION 3.1. *Let i be a task and $t \geq 0$ be any time. Consider the total size of jobs generated by i with deadlines at most t. Then, if $t < d_i$, the total size is 0; otherwise, i.e., $t \geq d_i$, the total size is $c_i + \left\lfloor \frac{t-d_i}{p_i} \right\rfloor c_i \leq c_i + \frac{c_i}{p_i} \cdot t$.*

3.2.1 Tasks with general weights.
For tasks with general weights, our algorithm makes use of a result for the following problem.

> BUDGETED CONTINUOUS REAL-TIME SCHEDULING (BCRS). Given a set J of jobs, where each job $j_i \in J$ is associated with a time window $W_i = [r_i, d_i]$, a size c_i, a cost s_i and a weight w_i. A job j_i can be completed by processing it in W_i for c_i units of time nonpreemptively, i.e., for any

interval $[t, t + c_i] \subseteq [r_i, d_i]$. A subset $J' \subseteq J$ is *feasible* if all jobs in J' can be completed with non-overlapping processing intervals and the total cost of all jobs in J' is at most 1. The problem is to find a feasible set J' with maximum total weight.

Lemma 3.1. ([14]) *There is a $(4+\epsilon)$-approximate algorithm for* BCRS.

Our algorithm is defined as follows.

Algorithm 1 General weight algorithm.
```
1: For each task $i$ in the input set $I$, let $j_i$ be the first job generated by $i$. We let $W_i = [0, d_i]$ be the time window associated with $j_i$. We define the size of $j_i$ as $c_i$, the cost $s_i$ as $c_i/p_i$ and the weight as $w_i$. Let $J$ be the set of these first generated jobs $j_i$ over all tasks $i \in I$.
2: We invoke the algorithm of [14] with the set $J$. Let $J' \subseteq J$ be the feasible set of jobs returned.
3: Let $I'$ be the set of tasks corresponding to $J'$, i.e., $I' = \{i : j_i \in J'\}$. We return $I'$ as the output.
```

We prove the performance of the above algorithm by the following two lemmas. Let Opt be any feasible subset of tasks.

Lemma 3.2. *The set I' of tasks has total weight at least $1/(4+\epsilon)$ times that of* Opt.

Proof. Let $J'' \subseteq J$ be the set of first generated jobs corresponding to tasks in Opt. Since all jobs in J'' are released at time 0, they can be completed nonpreemptively by EDF. Furthermore, the total cost of J'', which is $\sum_{j_i \in J''} c_i/p_i$, is at most 1. Hence, J'' is a feasible set of jobs. Then, by Lemma 3.1, the set J'' has total weight at most $(4+\epsilon)$ times that of J'. Equivalently, I' has total weight at least $1/(4+\epsilon)$ times that of Opt. □

Hence, if I' can be scheduled on a speed-2 processor, we have a 2-speed $(4+\epsilon)$-approximate algorithm. To show this, we prove a more general lemma as follows.

Lemma 3.3. *Assume a set of tasks V satisfies the following two properties.*

1. *The set of first generated jobs over all tasks in V can be completed by a speed-x processor.*
2. *The total utilization of V, i.e., $\sum_{i \in V} c_i/p_i$ is at most y.*

Then, the jobs generated by V can be scheduled by a speed-$(x+y)$ processor by EDF.

Proof. Let t be any time. Consider the set of the jobs generated by tasks in V with deadlines at most t. We want to show that these jobs have total size at most $(x+y)t$. We observe that it will imply the feasibility of EDF by induction. Specifically, the implication is true when there is only one job. When there are $n > 1$ jobs, the induction hypothesis states that the $n-1$ jobs with earliest deadlines will be completed, while the latest deadline job will be completed because the total size of jobs up to its deadline d is at most $(x+y)d$.

It remains to prove the above size bound. Let $V' \subseteq V$ be the tasks with relative deadlines at least t. Note that only V' can contribute to the total size concerned. Consider the first jobs generated by tasks in V'. Property 1 ensures that they can be completed by a speed-x processor, so their total size is at most xt. Property 2 states that the total utilization of V' is at most y, i.e., $\sum_{i \in V'} c_i/p_i \leq y$. Hence, by Observation 3.1, the total size of jobs generated by tasks in V' with deadlines at most t is bounded by

$$\sum_{i \in V'} \left(c_i + \frac{c_i}{p_i} t \right) \leq xt + yt,$$

and the lemma follows. □

Theorem 3.3. *Algorithm 1 is 2-speed $(4+\epsilon)$-approximate for* MaxFS *in the synchronous single processor case.*

Proof. By Lemma 3.2, the total weight of I' is at least $1/(4+\epsilon)$ times that of Opt. Note that the corresponding set J' is feasible on a unit-speed processor and the total utilization of I' is at most 1. By Lemma 3.3, I' can be scheduled by a speed-2 processor and the theorem follows. □

3.2.2 Tasks with weight equal to utilization.

When the weight of each task i equals its utilization, i.e., $w_i = c_i/p_i$, we give a better algorithm as follows.

Algorithm 2 Algorithm for weights = utilization
```
1: Define the set $J$ of jobs as in Step 1 of Algorithm 1. Note that each job $j_i$ has cost $s_i = w_i = c_i/p_i$.
2: Find a set $J' \subseteq J$ of jobs with the following two properties.
     • The total weight of $J'$ is at least the total weight of any feasible set $J'' \subseteq J$.
     • $J'$ can be completed by a speed-2 processor and has total cost at most 2.
3: Define the set $I'$ of tasks as in Step 3 of Algorithm 1. Return $I'$ as the output.
```

We can perform Step 2 by the following greedy algorithm. We define the *density* of a job j_i to be its weight divided by its size, i.e., density equals w_i/c_i. Initially, let J' be an empty set. We consider each job in J in descending order of density. When considering a job j_i, we check whether $J' \cup j_i$ can be completed by a speed-2 processor. If yes, we include j_i into J', i.e., setting $J' = J' \cup j_i$. Otherwise, we discard j_i. We then stop if the total cost of J' is at least 1, or terminate after all jobs are considered.

LEMMA 3.4. *The set J' returned by the above greedy algorithm satisfies the two properties in Step 2.*

THEOREM 3.4. *Algorithm 2 is 4-speed 1-approximate for* MAXFS *in the synchronous single processor case when for every task the weight equals the utilization.*

4 Open Problems

Several interesting open problems remain in the context of this paper.

1. Is there a polynomial time algorithm for 1-APPROXIMATE FEASIBILITY in non-synchronous multiprocessor systems with a fixed number of task types?
2. Is there a constant-speed, constant-approximation algorithm for MAXFS in non-synchronous uniprocessor systems?
3. Can the results for MAXFS be extended to multiprocessor systems?

From a broader perspective, it would be interesting to determine other tractable special cases of the feasibility problem.

Acknowledgments. We thank Benjamin Doerr for his help with the derandomization of Theorem 2.5.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proc. 16th Euromicro Conf. on Real-Time Systems*, pages 187–195, 2004.

[2] E. Bach and J. Shallit. *Algorithmic number theory. Vol. I: Efficient algorithms.* MIT Press, 1996.

[3] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2001.

[4] S. K. Baruah and T. P. Baker. Schedulability analysis of global EDF. *Real-Time Systems*, 38(3):223–235, 2008.

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

[6] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. In D. Halperin and K. Mehlhorn, editors, *Proc. 16th European Symp. on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 210–221. Springer, 2008.

[7] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *Proc. 23rd Real-Time Systems Symp.*, pages 159–168, 2002.

[8] B. Doerr. Private communication, 2009.

[9] F. Eisenbrand and T. Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proc. 21st ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[10] K. M. Elbassioni, R. Raman, S. Ray, and R. Sitters. On the approximability of the maximum feasible subsystem problem with 0/1-coefficients. In *Proc. of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 1210–1219, 2009.

[11] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 53(3):324–360, 2006.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[13] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[14] A. Kulik and H. Shachnai. On Lagrangian relaxation and subset selection problems. In *Proc. 6th Workshop on Approximation and Online Algorithms*, pages 160–173, 2009.

[15] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.

[16] E. L. Lawler and C. U. Martel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters*, 12(1):9–12, 1981.

[17] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[18] J. Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115–118, 1980.

[19] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[20] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[21] M. Mignotte. How to share a secret? In *Proc. of the Workshop on Cryptography*, pages 371–375, 1982.

[22] I. Niven, H. Zuckerman, and H. Montgomery. *An introduction to the theory of numbers.* John Wiley & Sons, 1991.

[23] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley Publishing Company, Reading, MA, 1994.

[24] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[25] B. Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63(1):211–232, 1941.

[26] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

A Construction of a Mignotte sequence

LEMMA A.1. *Let $\pi([a,b])$ denote the number of primes in the interval $[a,b]$. For sufficiently large x, $\pi[x, \sqrt[3]{2}x] \geq x/(4\ln(x))$.*

Proof. For $x \geq 55$, the number of primes up to x, $\pi(x)$, is bounded [25] by

$$\frac{x}{\ln x + 2} \; < \; \pi(x) \; < \; \frac{x}{\ln x - 4}\,.$$

For x sufficiently large, this gives

$$\begin{aligned}\pi([x, \sqrt[3]{2}x]) &= \pi([0, \sqrt[3]{2}x]) - (\pi[0,x]) \\ &\geq \frac{\sqrt[3]{2}x}{\ln(\sqrt[3]{2}x) + 2} - \frac{x}{\ln x - 4} \\ &= \frac{x}{\ln x}\left(\frac{\sqrt[3]{2}}{1 + \frac{2+\ln\sqrt[3]{2}}{\ln x}} - \frac{1}{1 - \frac{4}{\ln x}}\right) \\ &\geq \frac{x}{4\ln x}\,.\end{aligned}$$

□

LEMMA A.2. ([8]) *A (k,n)-Mignotte sequence can be constructed in time that is polynomial in n.*

Proof. It is sufficient to construct an arbitrary set S of $k \leq n$ pairwise coprime integers in $[2^n, 2^{n+1}]$ since every such set is a (k,n)-Mignotte sequence.

We identify $k-1$ pairs a_i, b_i of primes with $a_i \in [n^2, \sqrt[3]{2}n^2]$ and $b_i \in [\sqrt[3]{2}^2 n^2, 2n^2]$, for $i = 1, \ldots, k-1$. By Lemma A.1 there are sufficiently many primes in both intervals if n is large enough. Thus, we can simply search both intervals and test primality in time that is polynomial in n [22]. Notice that $\sqrt[3]{2} \leq b_i/a_i \leq 2$.

Now we are ready to find the desired set of coprimes starting from $S = \{2^n\}$. In each iteration i, for $i = 1, \ldots, k-1$, find a candidate which is a power of the prime a_i as follows. Let $x = \lfloor \log_{a_i} 2^{n+1} \rfloor$. If the candidate a_i^x lies in the interval, then add it to S. Otherwise (that is when $a_i^x < 2^n$), multiply the candidate sufficiently often with b_i/a_i until it lies in the desired interval.

Clearly, all elements $s \in S$ are by construction pairwise coprime. It is left to show that every candidate $a_i^x < 2^n$ needs to be multiplied by factor b_i/a_i at most x times, which is polynomial in n, until it certainly exceeds 2^n. In that case, it must lie in the interval since the factor is at most 2.

Let y be the parameter in question, i.e., y is the smallest exponent such that $a^x(b/a)^y \geq 2^n$, where $a := a_i$ and $b := b_i$. Clearly, $y = \lceil \log_{b/a} 2^n/a^x \rceil$. By definition, we have $a^{x+1} > 2^{n+1}$ which implies $2^n/a^x < a/2$. Hence,

$$\begin{aligned} y = \left\lceil \log_{b/a} \frac{2^n}{a^x} \right\rceil &< \log_{\sqrt[3]{2}} \frac{a}{2} + 1 \; < \; \log_{\sqrt[3]{2}} n^2 + 1 \\ &= 2\log_{\sqrt[3]{2}} n + 2\,. \end{aligned}$$

The desired bound, $y \leq x = \lfloor \log_{a_i} 2^{n+1} \rfloor$, follows for sufficiently large n. □