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

Keywords: real-time scheduling, periodic task system, feasibility test, Earliest Deadline First, approximation algorithms, computational complexity, inapproximability

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 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 migrations, so as to meet its deadline.

In a *periodic task system* T, a task $i \in T$ 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 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 receive c_i time units of execution 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 assumption is made that $d_i \leq p_i$, for all $i \in T$. We also assume all input parameters to have integer value; rational values can also be accomodated, by clearing denominators. Execution of a job can be stopped at any time and resumed later on a different processor, without penalty.

A task system is said to be *feasible* if there exists a schedule in which 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 preliminary version of this work appeared in *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2010.

†{bonifaci,nmegow}@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

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

§alberto@dis.uniroma1.it. Sapienza Università di Roma, Italy. Partially supported by the ICT Programme of the European Union under contract ICT-2008-215270 (FRONTS).

A well-known necessary condition for the feasibility of a task system T on m processors is that $U(T) := \sum_{i \in T} c_i/p_i \leq m$. The quantity $U(T)$ is called the *utilization* of the task system and c_i/p_i is called the utilization of task i. However, the condition $U(T) \leq m$ is far from being sufficient for feasibility. In fact, the feasibility problem for periodic task systems is coNP-hard [7, 23].

In the hope of overcoming hardness results, it is meaningful to relax the accuracy requirements of the feasibility problem slightly. For this reason, the concept of approximate feasibility has been introduced [9], which can be interpreted as a form of *resource augmentation* [17, 29]. For a fixed *speedup* parameter $\sigma \geq 1$, the problem of deciding σ-approximate feasibility is as follows.

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

- YES implies that T is feasible on m speed-σ processors, and
- NO implies that T 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 T, a positive integer m, weights $w : T \to \mathbb{Q}_+$.
Output: a subset of tasks $S \subseteq T$ 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. On the other hand, an approximate solution to the weight maximization problem does not immediately yield a useful answer to the feasibility problem, so MaxFS might be easier from the point of view of approximate solutions.

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: intuitively, the reason is that the first missed deadline might occur after an exponential amount of time [7, 23].

In the special case of uniprocessor scheduling with a constant number of distinct task types, Baruah et al. [7] 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 $i \in T$. In this setting, Albers and Slomka [1] provide a polynomial time $(1 + \epsilon)$-approximate feasibility test on a single processor, for any $\epsilon > 0$. A pseudopolynomial time feasibility test is possible when $U(T) \leq \mu$ for some constant $\mu < 1$ [7]. The complexity of the exact – that is, 1-approximate – feasibility problem for synchronous task systems has been open for a long time [7]. Independently of our work, Eisenbrand and Rothvoss [11] showed that this problem is weakly coNP-hard already in the uniprocessor case.

In the multiprocessor case, the feasibility problem seems even harder. The best algorithm known uses exponential time *and* space [21]. Phillips et al. [29] 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, again, this does not yield an efficient test for feasibility, or even approximate feasibility. However, recently some approximate feasibility tests have been derived for *sporadic* task systems [6,8]. 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 a natural extension of the feasibility problem that is relevant in various applications, which is also reflected by the attention that related scheduling problems received in the past, see for example [5, 12, 19, 20] and references therein. The crucial difference between previous considerations and our setting lies in the periodicity of the tasks. We are not aware of any existing result on weight maximization 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 $\epsilon > 0$, even on a single processor. A similar argument also shows that σ-Approximate Feasibility is *strongly* coNP-hard for any constant σ. Assuming P≠NP, this rules out any polynomial or pseudopolynomial time algorithm for testing feasibility within a constant speedup factor. Since augmenting the speed is equivalent to shrinking the execution times, a consequence is that the feasibility problem remains coNP-hard even for task systems with utilization bounded by an arbitrarily small constant. This contrasts with previous positive approximability results for sporadic task systems [1, 6, 8].

To solve the complexity status of σ-Approximate Feasibility, we reduce from a maximization variant of the number theoretic Simultaneous Congruences problem; see for example [24]. 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 the feasibility problem for multiple processors is coNP-hard. To this aim we first define and study Least Common Multiple Packing, a number theoretic problem that given a set T of integers and two integers k and L requires to find a subset $S \subseteq T$ of cardinality at least k, such that the least common multiple of integers in S is no more than L; we then give a reduction from Least Common Multiple Packing to the feasibility problem. Independently of our work, Eisenbrand and Rothvoss proved that even the uniprocessor case of the feasibility problem is coNP-hard for synchronous systems [11]. Thus, our result is narrower in scope than the one in [11], since it applies only to multiprocessor systems. However, our proof of the result is completely different and we believe it might be of independent interest.

We complement our negative results for arbitrary periodic tasks with the first constant approximation algorithms for two restricted models. We provide a polynomial time $(2-1/m)$-approximate test for multiprocessor task systems with a constant number of different task types. Similar to the uniprocessor test by [7], we decide feasibility by solving integer linear programs (ILPs) of constant dimension; in our case, however, solving a single ILP is not sufficient and we need to consider a constant number of them. For synchronous 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 this positive result, we introduce a refinement of a notion of total workload per interval, which was introduced recently in the context of sporadic task systems [8].

We already mentioned that MaxFS is not easier than the problem of deciding the feasibility of a task system. We show that MaxFS is NP-hard to approximate within $n^{1-\epsilon}$, even in the case of a uniprocessor and of unit task weights. Moreover, we show that MaxFS is NP-hard even in the strongly restricted setting of synchronous arrivals with *implicit deadlines*, where $d_i = p_i$ for all tasks. On the positive side, we give the first constant-speed, constant-approximation algorithm for synchronous task systems: a $(3 - 1/m)$-speed ρ_m-approximate algorithm, where $\rho_m = 3 + \epsilon$ for $m = 1$ and $\rho_m = 8 + \epsilon$ for $m > 1$.

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

	Single processor			Multiple processors		
	σ	Complexity		σ	Complexity	
Arbitrary systems	$n^{1-\epsilon}$	coNP-complete	$*$			
	1	coNP-complete	[7]	1	PSPACE	$*$
Synchronous systems	$1+\epsilon$	P	[1]	$2-1/m+\epsilon$	P	$*$
	1	coNP-complete	[11]	1	coNP-hard	$*$
Constant no. of task types	1	P	[7]	$2-1/m$	P	$*$
				1	pseudopoly	$*$

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

	Single processor				Multiple processors			
	σ	ρ	Complexity		σ	ρ	Complexity	
Arbitrary systems	1	$n^{1-\epsilon}$	NP-hard	$*$				
	$n^{1-\epsilon}$	1	coNP-hard	$*$				
Synchronous systems	2	$3+\epsilon$	P	$*$	$3-1/m$	$8+\epsilon$	P	$*$
	1	1	NP-hard	$*$				

Table 2: Results for Maximum Weight Feasible Subsystem. The notation is as in Table 1, where in addition ρ is the approximation factor.

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 [23] reduce from the Simultaneous Congruences problem. This problem is known to be NP-complete, even in the strong sense [7, 24]. We consider the following natural maximization variant of the decision problem.

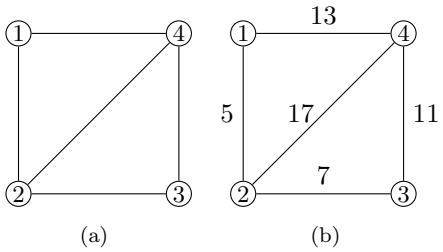
Maximum Simultaneous Congruences (MaxSC)
Input: $a_1, \ldots, a_n \in \mathbb{N}$, $b_1, \ldots, b_n \in \mathbb{N}$.
Output: $S \subseteq \{1, \ldots, n\}$ such that the set $\{t \in \mathbb{N} : t \equiv a_i \pmod{b_i} \text{ for all } i \in S\}$ is nonempty.
Objective: Maximize $|S|$.

This problem can be seen as a Maximum Feasible Subsystem type of problem [3], with univariate congruences in place of multivariate linear equalities. We show the following inapproximability result for MaxSC.

Lemma 2.1. *For any $\epsilon > 0$,* MaxSC *is* NP*-hard to approximate 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 $n^{1-\epsilon}$ [33]. Consider a graph $G(V, E)$ where $V = \{1, 2, \ldots, n\}$. We set $a_i = i$ for $i \in V$. Moreover, to every edge $e \in E$ we associate a distinct prime number $\pi(e) > n$. We remark that this step can be implemented in polynomial time, since for example it is known [28] that there are at least n^2 prime numbers in the range $(n, 4n^4)$, and we can find them by an exhaustive search. For every node $i \in V$ with the set of incident edges $\delta(i)$ we define $b_i := \prod_{e \in \delta(i)} \pi(e)$; see Figure 1.

Now if $(i, j) \notin E$ then $\gcd(b_i, b_j) = 1$ and so $a_i \equiv a_j \pmod{\gcd(b_i, b_j)}$. If $(i, j) \in E$ then $\gcd(b_i, b_j) = \pi((i, j)) > \max(a_i, a_j)$ so that $a_i \not\equiv a_j \pmod{\gcd(b_i, b_j)}$, simply because $a_i \neq a_j$, and so the two congruences $t \equiv a_i \pmod{b_i}$, $t \equiv a_j \pmod{b_j}$ cannot have simultaneous solution.



a_1	1	b_1	$5 \cdot 13 = 65$	$t \equiv 1 \pmod{65}$
a_2	2	b_2	$5 \cdot 7 \cdot 17 = 595$	$t \equiv 2 \pmod{595}$
a_3	3	b_3	$7 \cdot 11 = 77$	$t \equiv 3 \pmod{77}$
a_4	4	b_4	$11 \cdot 13 \cdot 17 = 2431$	$t \equiv 4 \pmod{2431}$

(c)

Figure 1: The reduction from Max Independent Set to Max Simultaneous Congruences. (a) Original graph; (b) prime numbers associated to the edges of the graph; (c) corresponding system of congruences.

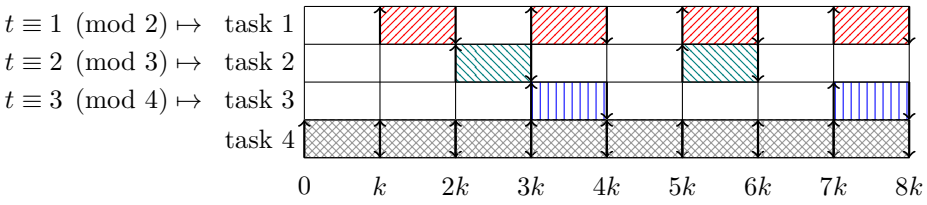


Figure 2: The reduction from Max Simultaneous Congruences to σ-Approximate Feasibility.

Thus, by the Generalized Chinese Remainder Theorem, see for example [4], a set S of congruences is satisfiable if and only if S is an independent set in G. The claim follows. □

Theorem 2.2. *For any $\epsilon > 0$ and $1 \leq \sigma \leq n^{1-\epsilon}$, 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 congruence 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 Lemma 2.1.

We associate a task to every congruence. For each $1 \leq i \leq n$, 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 $d_{n+1} = p_{n+1} = k$; see also Figure 2. 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 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. □

We observe that the numbers encoded in the reductions above are in general exponentially large; one could then wonder if allowing a pseudopolynomial running time can improve the approximation

ratio. This turns out not to be the case.

Theorem 2.3. *For any constant $\sigma \geq 1$, σ-*Approximate Feasibility *is strongly* coNP*-hard, even in the single processor case.*

Proof. It will be enough to show that it is strongly NP-hard to approximate MaxSC within a factor of σ; the result then follows by the same argument as in Theorem 2.2. We use the same construction as in Lemma 2.1, except that we reduce from instances of Maximum Independent Set in which the degree of the graph is bounded by some constant Δ. It is known that there is some $\epsilon > 0$ such that this problem is NP-hard to approximate within Δ^ϵ [2]. Pick the smallest Δ such that $\Delta > \sigma^{1/\epsilon}$. For any fixed σ, this yields a constant bound on the degree of the graph and so the numerical values (the a_i's and b_i's) constructed in the reduction of Lemma 2.1 are polynomially bounded in n. An approximation algorithm with a ratio of σ for MaxSC would imply that Maximum Independent Set can be approximated within a factor of Δ^ϵ, which is strongly NP-hard. □

For a single processor, the feasibility problem is always in coNP – the existence of short witnesses of infeasibility has been known since quite some time [7]. However, that is not known to hold for the multiprocessor case. We conclude this section by observing that the multiprocessor case can at least be solved in polynomial space. We need the following definition.

Definition 2.1. A schedule is called *cyclic* if the following holds for each processor and each time $t \geq \max_i r_i$: if the processor is idle at time t, then it is idle at time $t + \text{lcm}\{p_1, \ldots, p_n\}$, and if the processor is working on a job of task i at time t, then it is working on another job of task i, released $\text{lcm}\{p_1, \ldots, p_n\}$ time units later, at time $t + \text{lcm}\{p_1, \ldots, p_n\}$.

It is an old result that it suffices to consider cyclic schedules to determine feasibility.

Proposition 2.4 ([21])**.** *A periodic task system T is feasible if and only if it admits a cyclic schedule.*

Theorem 2.5. *The feasibility problem for periodic task systems is in* PSPACE.

Proof. It is enough to prove the existence of a nondeterministic polynomial space algorithm for the feasibility problem, since nondeterminism can always be removed at the cost of squaring the amount of space required [32]. By Proposition 2.4, it is enough to decide whether a cyclic schedule exists. We can take as a reference the interval $[t_{\min}, t_{\max}] = [\max_i r_i, \max_i r_i + \text{lcm}\{p_1, \ldots, p_n\}]$. Let j_i be a generic job from task i; denote its release date by $r(j_i)$ and its absolute deadline by $d(j_i)$. If j_i is such that $r(j_i) \in [t_{\min}, t_{\max}]$, but $d(j_i) > t_{\max}$, we "wrap" j_i around – in other words we make it available for processing in both the intervals $[r(j_i), t_{\max}]$ and $[t_{\min}, d(j_i) - \text{lcm}\{p_1, \ldots, p_n\}]$. Otherwise its availability window is simply $[r(j_i), d(j_i)]$. Notice that all these intervals have integral extreme points; thus, it suffices to restrict to schedules that preempt and migrate only at integral time points (see [7] for a proof of this fact).

In order to keep track of the jobs' availability windows it is sufficient to keep one "global clock" counter of polynomially many bits, since $t_{\max}$ is at most exponentially large in the input size. Moreover, we keep one counter for each task i that counts how much execution the currently active job from task i (if any) has already received; the assumption that $d_i \leq t_i$ implies that at most one job from each task can be pending at any time. Another counter for each task is sufficient to track the processing of wrapped-around jobs, since there is at most one such job for each task. The algorithm now guesses, at each time step, the set of at most m jobs to be scheduled, and updates the counters accordingly. If at any time some deadline is missed, we report a failure; otherwise we report success after reaching time $t_{\max}$. Some nondeterministic execution of this algorithm succeeds if and only if the task system is feasible. □

2.2 Task systems with a constant number of task types

We have seen that Theorem 2.2 excludes the existence of any constant-approximate polynomial time algorithm for deciding the feasibility of an arbitrary periodic task system. However, for the

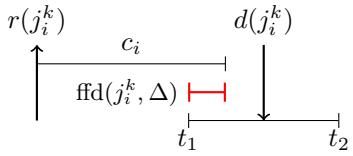


Figure 3: The forward forced demand of a job.

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. [8] introduced a lower bound on the total processing requirement of a task system in an interval, which they called *forward forced demand* (ffd). (In the following we use the shorthand $x^+ := \max\{x, 0\}$).

Definition 2.2 (Forward forced demand)**.** Consider a task system T where a task $i \in T$ consists of jobs j_i^k, $k = 1, 2, \ldots$, with corresponding release dates $r(j_i^k) := r_i + (k-1)p_i$ and deadlines $d(j_i^k) := r_i + (k-1)p_i + d_i$. Given a time interval $\Delta = [t_1, t_2]$, we define

$$\begin{aligned}
\mathrm{len}(\Delta) &:= t_2 - t_1 \\
\mathrm{ffd}_T(j_i^k, \Delta) &:= \begin{cases} (c_i - (t_1 - r(j_i^k))^+)^+ & \text{if } d(j_i^k) \in \Delta, \\ 0 & \text{otherwise;} \end{cases} \\
\mathrm{ffd}_T(i, \Delta) &:= \sum_{k \in \mathbb{N}} \mathrm{ffd}_T(j_i^k, \Delta) \\
\mathrm{ffd}_T(\Delta) &:= \sum_{i \in T} \mathrm{ffd}_T(i, \Delta).
\end{aligned}$$

The definition is illustrated in Figure 3. Let k_i be the number of jobs of task i that are released strictly before t_1 and due within the interval Δ, and let k_i' be the number of jobs of task i that are released and due in Δ. Then a straightforward calculation gives

$$\mathrm{ffd}_T(\Delta) \;=\; \sum_{i \in T} k_i' c_i + (c_i - (t_1 - r_i - (k_i - 1)p_i)^+)^+. \tag{1}$$

Since the forward forced demand is a lower bound on the amount of work that has to be spent in a given time interval, the following necessary condition for feasibility holds.

Proposition 2.6. *If a periodic task system T is feasible on m unit speed processors, then* $\mathrm{ffd}_T(\Delta) \le m \cdot \mathrm{len}(\Delta)$ *for any interval Δ.*

The following result shows that a small forward forced demand is sufficient to ensure the EDF-schedulability of a task system on multiple processors of an appropriate speed. The claim was originally proved for sporadic task systems, but in fact it applies to arbitrary collections of jobs, and thus also to periodic task systems.

Lemma 2.7 ([8])**.** *If a periodic task system T is not* EDF*-schedulable on m speed-σ processors, then there is an interval Δ such that* $\mathrm{ffd}_T(\Delta)/\mathrm{len}(\Delta) > m(\sigma - 1) + 1$.

With these prerequisites we can state our result.

Theorem 2.8. *For periodic task systems with a constant number of task types and m processors, there is a polynomial time algorithm solving σ-*Approximate Feasibility*, for any $\sigma \ge 2 - 1/m$.*

Proof. Let s denote the number of distinct types of tasks each defined by a quadruple (r_i, c_i, d_i, p_i), and let n_i, for $i = 1, \ldots, s$, denote the number of tasks of the i-th task type. Furthermore, we use $\operatorname{lcm}\{p_1, \ldots, p_s\}$ to denote the least common multiple of periods $p_1, \ldots, p_s$. Assume there is an interval $\Delta = [t_1, t_2]$ such that $\operatorname{ffd}_T(\Delta) > m \cdot \operatorname{len}(\Delta)$. Without loss of generality we can assume that $r_i \leq t_1$ for each task $i \in T$; if not, we can increase both t_1 and t_2 by some multiple of $\operatorname{lcm}\{p_1, \ldots, p_s\}$ and the forward forced demand will not decrease.

We construct a system of linear and non-linear inequalities that characterizes such an interval Δ. By Proposition 2.6, a feasible solution of this system implies that T is infeasible.

$$r_i + p_i k_i \geq t_1, \qquad i = 1, \ldots, s \tag{2}$$

$$r_i + p_i(k_i - 1) < t_1, \qquad i = 1, \ldots, s \tag{3}$$

$$r_i + p_i k_i + p_i(k_i' - 1) + d_i \leq t_2, \qquad i = 1, \ldots, s \tag{4}$$

$$r_i \leq t_1, \qquad i = 1, \ldots, s \tag{5}$$

$$\sum_{i=1}^{s} n_i c_i k_i' + n_i (c_i - (t_1 - r_i - p_i(k_i - 1))^+)^+ > \; m(t_2 - t_1) \tag{6}$$

$$t_1, t_2, k_i, k_i' \in \mathbb{Z}_+.$$

The variables of this system of inequalities are t_1, t_2, and k_i, k_i', for $i = 1, \ldots, s$. Here, k_i is the number of jobs of a task of type i that are released strictly before t_1 – this is ensured by (2) and (3). Variable k_i' is the number of jobs of a task of type i that are released and due within the interval $[t_1, t_2]$, see (4). The left hand side of inequality (6) expresses $\operatorname{ffd}_T(\Delta)$ (compare with (1)), so (6) enforces that the workload inequality in Proposition 2.6 is violated, that is, $\operatorname{ffd}_T(\Delta) > m \cdot \operatorname{len}(\Delta)$.

All the inequalities are linear except the last one. The expression of $\operatorname{ffd}_T(\Delta)$ on the left hand side of inequality (6) contains the non-linear term $g_i := (c_i - (t_1 - r_i - p_i(k_i - 1))^+)^+$. Notice that by constraint (3) g_i can take only one of two values for any $i = 1, \ldots, s$:

$$g_i = \begin{cases} c_i - (t_1 - r_i - p_i(k_i - 1)) & \text{if } c_i - (t_1 - r_i - p_i(k_i - 1)) > 0 \quad (6') \\ 0 & \text{if } c_i - (t_1 - r_i - p_i(k_i - 1)) \leq 0 \quad (6''). \end{cases}$$

The idea now is to guess, for each i, which of the two cases occurs. That is, we consider 2^s integer linear programs. Every such program consists of the constraints (2)–(6) above, with inequality (6) simplified in the appropriate way, plus inequality (6') or (6") for each i, depending on the guess for the corresponding g_i term.

For any choice of g_i, for $i = 1, \ldots, s$, this yields a system of $5s + 1$ linear inequalities. Since s 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, using Lenstra's algorithm [22].

If any of these integer programs has a feasible solution, then we have found an overloaded interval Δ which witnesses that the task system is infeasible by Proposition 2.6. Otherwise, such an interval cannot exist and thus Lemma 2.7 implies that EDF yields a feasible schedule on m processors of speed $2 - 1/m$. □

When $m = 1$ the above test is exact, since $2 - 1/m = 1$. We do not know whether an exact feasibility test with polynomial running time is possible when $m > 1$. However, a simple pseudopolynomial time test does exist; in fact, it exists even with the weaker assumption that the number of distinct periods is constant.

Theorem 2.9. *For periodic task systems with a constant number of distinct periods the feasibility problem can be solved in pseudopolynomial time.*

Proof. Let k denote the number of distinct periods, and let L denote the least common multiple of the periods. Notice that $L \leq (\max_i p_i)^k$, which is pseudopolynomially large for fixed k.

As in the proof of Theorem 2.5, it is enough to show that a cyclic schedule exists for the interval $[t_{\min}, t_{\max}]$, with $t_{\min} = \max_i r_i$, $t_{\max} = \max_i r_i + L$. To this end we can use a standard

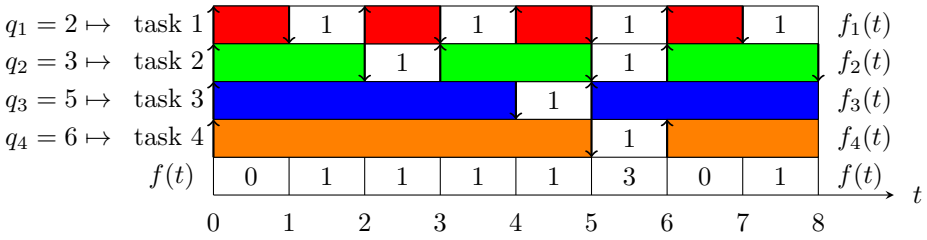


Figure 4: Construction used in the proof of Theorem 2.11.

construction [16] in which we formulate the feasibility problem for the finite set of jobs in $[t_{\min}, t_{\max}]$ as a maximum flow problem on a bipartite network, with one layer of nodes corresponding to time units (these nodes have a maximum inflow of m) and another layer of nodes corresponding to jobs (with a maximum outflow equal to the processing time), with a job connected to a time unit, via a unit capacity arc, if the job is available for processing in that time unit. A cyclic schedule exists if and only if a flow of value equal to the total processing requirement of the jobs exists in this network, which can be tested in pseudopolynomial time. □

2.3 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 $\operatorname{lcm}\{q_i : i \in S\} \leq L$?

Theorem 2.10. Least Common Multiple Packing *is* NP*-hard.*

Proof. A (k, n)*-Mignotte sequence* [26] 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 $\Pi_{1 \leq i \leq k} \pi_i > \Pi_{1 \leq i \leq k-1} \pi_{n-i+1}$. Such a sequence can be constructed in polynomial time by using the fact that for x being large enough each interval $[x, x + x^{3/5})$ contains a prime number [15]. Starting from $x_0 = n^{10}$, we construct a sequence of intervals $[x_{i-1}, x_i), i = 1, \ldots, n$, with $x_i := n^{10} + 2in^6 > x_{i-1} + x_{i-1}^{3/5}$, each of which is guaranteed to contain a prime number. Thus, the full interval $[n^{10}, n^{10} + n^8]$ contains n primes which can be found by exhaustive search. They form a (k, n)-Mignotte sequence, since $n^{10k} > (n^{10} + n^8)^{k-1}$ for n larger than some constant.

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 $L := \Pi_{1 \leq i \leq s} \pi_{n-i+1}$ and $k := \binom{s}{2} - 1$.

Now, if G has an s-clique, and S is the corresponding set of $k+1$ edges, we have $\operatorname{lcm}\{q_i : i \in S\} \leq \Pi_{1 \leq i \leq s} \pi_{n-i+1} = L$, since S spans exactly s vertices. 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 $\operatorname{lcm}\{q_i : i \in S\} \geq \Pi_{1 \leq i \leq s+1} \pi_i > L$. □

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

Theorem 2.11. *The feasibility problem for synchronous task systems is* coNP*-hard.*

Proof. We reduce from Least Common Multiple Packing. Given $q_1, \ldots, q_m, k, L \in \mathbb{N}$ we create a system of $m + k$ tasks. For $1 \leq i \leq m$, 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

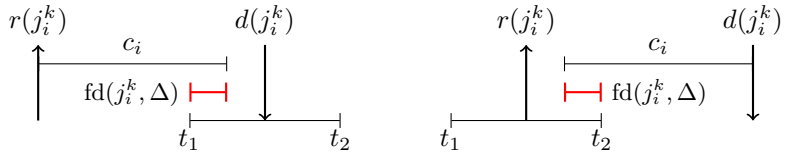


Figure 5: The forced demand of a job.

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 $t \geq 0$ and $1 \leq i \leq m$, let

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

That is, $f_i(t) = 1$ if and only if task i does *not* have to be scheduled during interval $[t, t+1]$. Furthermore, let $f(t) := \sum_{1 \leq i \leq m} f_i(t)$; this is the total number of "free" processor slots during $[t, t+1]$; see Figure 4 for an illustration. We now define the remaining k identical tasks by setting, for each $j = m+1, \ldots, m+k$: $r_j = 0$, $c_j = (1/k) \cdot \sum_{0 \leq t < L} f(t)$, $d_j = L$, $p_j = \mathrm{lcm}\{q_1, \ldots, q_m\}$. We remark that all these parameters can be computed in polynomial time, in particular $c_j = (1/k) \sum_{1 \leq i \leq m} \lfloor L/p_i \rfloor$.

For the analysis, consider the quantity $F := \max_{0 \leq t < L} f(t)$. 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 $\sum_{0 \leq t < L} f(t)$. However, because there are only k additional tasks and we cannot process a task simultaneously on more than one processor, the total useful time is in fact $\sum_{0 \leq t < L} \min(f(t), k)$. So it will be possible to schedule all the tasks if and only if $F \leq k$.

For a set $S \subseteq \{1, \ldots, m\}$, the minimum t for which $f_i(t) = 1$ for all $i \in S$ is easily seen to be $\mathrm{lcm}\{q_i : i \in S\} - 1$. Thus, $F \leq k$ if and only if there is no set S such that $|S| > k$ and $\mathrm{lcm}\{q_i : i \in S\} - 1 < L$, that is, if and only if the instance of Least Common Multiple Packing has no solution. Thus, coNP-hardness follows from Lemma 2.10. □

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 2.2). 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 j_i^k with deadline in $(t_2, t_2 + c_i)$ that might need to be partially scheduled also within $[t_1, t_2]$ to ensure feasibility. Motivated by this, we introduce a refinement of the forward forced demand.

Definition 2.3 (Forced demand)**.** Consider a set of tasks T where a task $i \in T$ consists of a finite or countable set of jobs j_i^k, $k = 1, 2, \ldots$, with corresponding release dates $r(j_i^k) := r_i + (k-1)p_i$ and deadlines $d(j_i^k) := r_i + (k-1)p_i + d_i$. Given an interval $\Delta = [t_1, t_2]$, we define

$$\mathrm{fd}_T(j_i^k, \Delta) := (c_i - (t_1 - r(j_i^k))^+ - (d(j_i^k) - t_2)^+)^+,$$

$$\mathrm{fd}_T(i, \Delta) := \sum_k \mathrm{fd}_T(j_i^k, \Delta),$$

$$\mathrm{fd}_T(\Delta) := \sum_{i \in T} \mathrm{fd}_T(i, \Delta).$$

Again, by construction, the forced demand of an interval is a lower bound on the total processing requirement of a feasible task system in that interval.

Proposition 2.12. *If a set of tasks T is feasible on m unit speed processors, then $\mathrm{fd}_T(\Delta) \leq m \cdot \mathrm{len}(\Delta)$ for any interval Δ.*

The following lemma shows that, in a synchronous system, $\mathrm{fd}(\Delta)/\mathrm{len}(\Delta)$ is maximized when the interval Δ starts at time 0; this is not necessarily the case for the ratio $\mathrm{ffd}(\Delta)/\mathrm{len}(\Delta)$.

Lemma 2.13. *For any synchronous task system T,*

$$\max_{\Delta} \frac{\mathrm{fd}_T(\Delta)}{\mathrm{len}(\Delta)} = \max_{t \in \mathbb{N}} \frac{\mathrm{fd}_T([0,t])}{t}.$$

Proof. Let $\Delta = [t_1, t_2]$ be such that $\mathrm{fd}_T(\Delta)/\mathrm{len}(\Delta)$ is maximized. We construct a new periodic (not necessarily synchronous) task system T' which differs from T in the start times and has no smaller forced demand: for each task i, choose a new release time $r'_i \in [0, p_i]$ such that a job of task i is released at t_1. To see that the forced demand does not decrease, we consider any task i and observe that the change in the fd value when increasing start times is due to (i) the decreased contribution of the last job j_i^ℓ released strictly before t_2 and (ii) the increased contribution of the last job j_i^k released strictly before t_1. No other job's contribution is affected. Now, (i) the decrease in the contribution of j_i^ℓ is bounded above by $\min\{ r'_i, c_i \}$, and (ii) the increased contribution of j_i^k is at least $\min\{ r'_i, c_i \}$. Thus, $\mathrm{fd}_{T'}(\Delta) \geq \mathrm{fd}_T(\Delta)$.

For periodic task systems this implies that the expression $\mathrm{fd}_T(\Delta)/\mathrm{len}(\Delta)$ is maximized on any interval of length $\mathrm{len}(\Delta)$ if all tasks simultaneously release a job at the beginning of the interval. By definition, in a synchronous system the interval $[0, t_2 - t_1]$ has exactly this property. Thus, $\mathrm{fd}_{T'}(\Delta) = \mathrm{fd}_T([0, t_2 - t_1])$, which implies the lemma. □

Lemma 2.14. *If a synchronous task system T is not* EDF*-schedulable on m speed-σ processors, then there is $t \in \mathbb{N}$ such that $\mathrm{fd}_T([0,t])/t > m(\sigma - 1) + 1$.*

Proof. By Lemma 2.7, if T is not EDF-schedulable, there is an interval Δ such that

$$m(\sigma - 1) + 1 < \mathrm{ffd}_T(\Delta)/\mathrm{len}(\Delta).$$

But $\mathrm{ffd}_T(\Delta)/\mathrm{len}(\Delta) \leq \mathrm{fd}_T(\Delta)/\mathrm{len}(\Delta) \leq \max_t \mathrm{fd}_T([0,t])/t$ by Lemma 2.13. The claim follows. □

Since by Lemma 2.13 we can focus on intervals of the form $[0,t]$, we obtain a simpler formula for the forced demand.

Proposition 2.15. *For any synchronous task system T and $t \in \mathbb{N}$,*

$$\begin{aligned} \mathrm{fd}_T([0,t]) &= \sum_{i \in T} \mathrm{fd}_T(i, [0,t]), \\ \mathrm{fd}_T(i, [0,t]) &= k_i c_i + (c_i - (k_i p_i + d_i - t)^+)^+, \\ \text{where } k_i &:= \left\lfloor \frac{t + p_i - d_i}{p_i} \right\rfloor. \end{aligned}$$

Figure 6 illustrates the function $\mathrm{fd}_T(i, [0,t])$.

Theorem 2.16. *Let $\epsilon > 0$. For synchronous task systems 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 approach is to approximate the *maximum load* of any time interval, that is, the quantity $\lambda^* := \max_t \mathrm{fd}_T([0,t])/t$. To this end we can adopt the same technique as in [8], of which we give here a streamlined proof. For each task i, define

$$\begin{aligned} \mathrm{thr}(i) &:= d_i - c_i + \lceil 1/\epsilon \rceil \cdot p_i, \\ \widehat{\mathrm{fd}}_T(i, [0,t]) &:= \begin{cases} \mathrm{fd}_T(i, [0,t]) & \text{if } t \leq \mathrm{thr}(i), \\ \frac{c_i}{p_i}\,(t - (d_i - c_i)) & \text{if } t > \mathrm{thr}(i). \end{cases} \end{aligned}$$

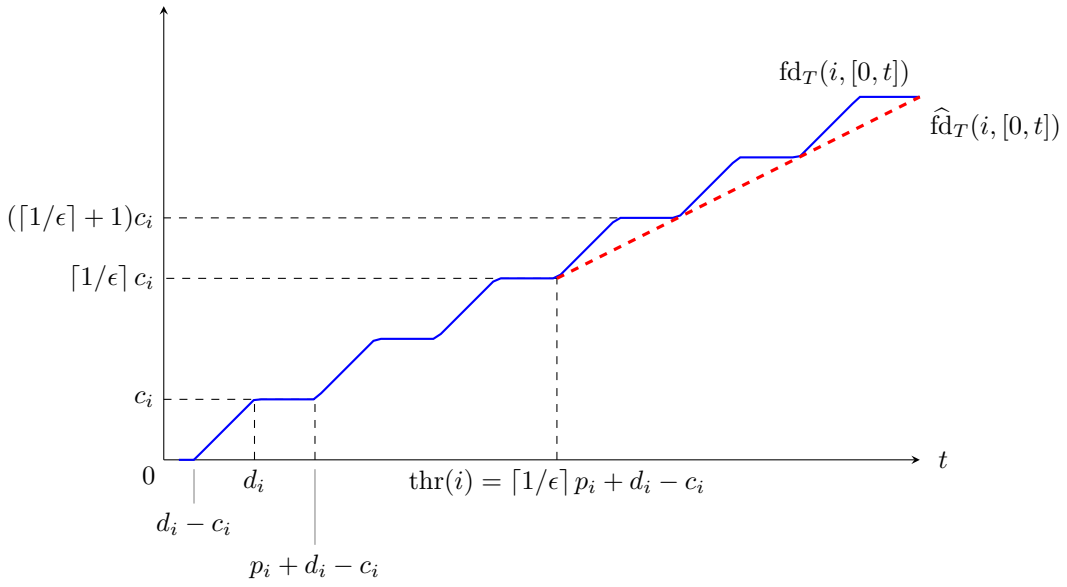


Figure 6: The function $\mathrm{fd}(i,[0,t])$ as a function of t (solid line) and its approximation $\widehat{\mathrm{fd}}(i,[0,t])$ (dashed line) used in the proof of Theorem 2.16.

The definition is illustrated in Figure 6. Notice that $\widehat{\mathrm{fd}}_T(i,[0,t]) \le \mathrm{fd}_T(i,[0,t]) \le (1+\epsilon)\widehat{\mathrm{fd}}_T(i,[0,t])$, since when $t \le \mathrm{thr}(i)$, $\widehat{\mathrm{fd}}_T(i,[0,t]) = \mathrm{fd}_T(i,[0,t])$, and when $t > \mathrm{thr}(i)$ we have

$$\begin{aligned}
\frac{\mathrm{fd}_T(i,[0,t])}{\widehat{\mathrm{fd}}_T(i,[0,t])} &\le \frac{\left\lfloor \frac{\mathrm{thr}(i)+p_i-d_i}{p_i} \right\rfloor c_i + c_i}{(c_i/p_i)(\mathrm{thr}(i) - (d_i - c_i))} \\
&= \frac{\left\lfloor \frac{\lceil 1/\epsilon \rceil p_i + p_i - c_i}{p_i} \right\rfloor c_i + c_i}{\lceil 1/\epsilon \rceil\, c_i} \\
&= \frac{\lceil 1/\epsilon \rceil + 1}{\lceil 1/\epsilon \rceil} \\
&\le 1 + \epsilon.
\end{aligned}$$

Summing across tasks we obtain

$$\widehat{\mathrm{fd}}_T([0,t]) \le \mathrm{fd}_T([0,t]) \le (1+\epsilon)\widehat{\mathrm{fd}}_T([0,t]) \qquad \text{for all } t \in \mathbb{N}. \tag{7}$$

The main observation is that $\widehat{\mathrm{fd}}_T([0,t])$ is a piecewise linear function with breakpoints in the set

$$\begin{aligned}
K = &\bigcup_{i \in T} \{t \le \mathrm{thr}(i) \,:\, \exists k \in \mathbb{N} \,:\, t = (k-1)\cdot p_i + d_i - c_i\} \\
\cup &\bigcup_{i \in T} \{t \le \mathrm{thr}(i) \,:\, \exists k \in \mathbb{N} \,:\, t = (k-1)\cdot p_i + d_i\}.
\end{aligned}$$

Consequently, the function $\widehat{\mathrm{fd}}_T([0,t])/t$ is piecewise monotone and achieves its maximum at a point in K. Since the cardinality of K is $\mathcal{O}(n/\epsilon)$, the maximum can be found efficiently. Let λ be its value, so that $\lambda \le \lambda^* \le (1+\epsilon)\lambda$ by (7). Now we compare λ with m: if $\lambda > m$, there must be an interval Δ such that $\mathrm{fd}_T(\Delta) > m \cdot \mathrm{len}(\Delta)$, and by Proposition 2.12 the task system cannot be feasible on m unit speed machines. If on the other hand $\lambda \le m$, then for any $t \in \mathbb{N}$,

$$\begin{aligned}
\frac{\mathrm{fd}_T([0,t])}{t} &\le \lambda^* \le (1+\epsilon)\lambda \\
&\le (1+\epsilon)m,
\end{aligned}$$

Algorithm 1 Approximate feasibility test for synchronous task systems

1: For each $i \in T$:

$$\begin{aligned} \text{thr}(i) &\leftarrow d_i - c_i + \lceil 1/\epsilon \rceil \cdot p_i, \\ K_i &\leftarrow \{t \leq \text{thr}(i) \ : \ \exists k \in \mathbb{N} \ : \ t = (k-1) \cdot p_i + d_i - c_i\} \\ &\quad \cup \{t \leq \text{thr}(i) \ : \ \exists k \in \mathbb{N} \ : \ t = (k-1) \cdot p_i + d_i\}. \end{aligned}$$

2: $K \leftarrow \bigcup_{i \in T} K_i$.
3: $\lambda \leftarrow \max_{t \in K} \text{fd}_T([0,t])/t$.
4: If $\lambda > m$ return "infeasible on m unit speed machines".
5: If $\lambda \leq m$ return "EDF-schedulable on m speed $(2 - 1/m + \epsilon)$ machines".

and by Lemma 2.14 (with $\sigma = 2 - 1/m + \epsilon$) the task system must be EDF-schedulable on m speed-$(2 - 1/m + \epsilon)$ machines. The resulting algorithm is summarized as Algorithm 1. □

The factor $2 - 1/m$ in the statement of Theorem 2.16 is tight when schedulability is witnessed by EDF, since there exist feasible job sets that cannot be scheduled by EDF unless the speed is augmented by at least $2 - 1/m$ [29]. Moreover, the ϵ error term cannot be removed when $m = 1$, unless P=NP, as that would imply an exact polynomial-time feasibility test, while the problem is coNP-hard [11].

3 The maximum weight feasible subsystem problem

3.1 Hardness

Theorem 3.1. *For any $\epsilon > 0$,* MaxFS *is* NP*-hard to approximate within a factor $n^{1-\epsilon}$, even in the single processor case with unit task weights.*

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 [13]. Using the same construction as in Lemma 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 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. 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 execution times are 1. □

Theorem 3.2. MaxFS *is* NP*-hard even in the synchronous, single processor case when $d_i = p_i$ for all tasks $i \in T$.*

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 $c_i = w_i = a_i$, $d_i = p_i = A$, $r_i = 0$ 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, $\sum_{i \in S} a_i \leq A$ [25]. Now an optimal subset of tasks has total weight A if and only if there is a subset S such that $\sum_{i \in S} a_i = A$. □

3.2 Approximation algorithm for synchronous systems

Theorem 3.1 motivates us to focus on synchronous systems, for which we give a $(3-1/m)$-speed ρ_m-approximate algorithm, where $\rho_m = 3+\epsilon$ for $m = 1$ and $\rho_m = 8+\epsilon$ for $m > 1$. Our algorithm will build on approximation algorithms for the following auxiliary problems.

Max-weight Preemptive Real-Time Scheduling (Prts)
Input: a set of jobs $J = \{j_i\}_{i \in T}$, each with release date r_i, execution time c_i, absolute deadline d_i, and weight w_i, as well as a positive integer m.
Output: a schedule of J on m machines, with preemption and migration allowed.
Objective: maximize the total weight of on-time jobs (a job is *on-time* if it is completed within the deadline).

Budgeted Max-Weight Preemptive Real-Time Scheduling (Bprts)
Input: same as in the Prts problem, and in addition a cost b_i for each job, and a budget B.
Output: a feasible subset $J' \subseteq J$ of jobs. A subset J' is *feasible* if all jobs in J' can be scheduled on-time on the m machines and the total cost of all jobs in J' is at most B.
Objective: maximize the total weight of J'.

Lemma 3.3 ([18,30]). *For $m = 1$, there is a $(1+\epsilon)$-approximate algorithm for* Prts. *For $m > 1$, there is a $(6+\epsilon)$-approximate algorithm for* Prts.

Proof. The first claim follows by a result of Pruhs and Woeginger [30, Theorem 4.4]: there exists a $(1+\epsilon)$-approximate algorithm for maximizing the weighted number of on-time jobs in the scheduling problem $1|pmtn, r_j|\sum_j w_j(1-U_j)$, which is exactly Prts when $m = 1$. The proof in [30] is in fact for the minimization version $1|pmtn, r_j|\sum_j w_j U_j$, but the same argument applies to the maximization variant.

The second claim follows by the first, combined with a result by Kalyanasundaram and Pruhs [18, Theorem 3.1]: if there is a ρ-approximate algorithm for $1|pmtn, r_j|\sum_j w_j(1-U_j)$, then there is a 6ρ-approximate algorithm for $P|pmtn, r_j|\sum_j w_j(1-U_j)$. □

Lemma 3.4. *There is a ρ_m-approximate algorithm for* Bprts, *with $\rho_m = 3+\epsilon$ for $m = 1$, and $\rho_m = 8+\epsilon$ for $m > 1$.*

Lemma 3.4 follows directly from Lemma 3.3 and the following fact on subset selection problems, proved by Kulik and Shachnai [19]. A *subset selection problem* is a maximization problem in which any subset of a feasible solution is also feasible. Notice that Bprts is a subset selection problem, and Prts is a relaxation of Bprts without the budget constraint.

Lemma 3.5 ([19]). *Given a subset selection problem with a linear budget constraint, if there is a ρ-approximate algorithm for the problem without the budget constraint, then for any $\epsilon > 0$ there is a $(\rho+2+\epsilon)$-approximate algorithm for the problem with the budget constraint.*

Our algorithm for MaxFS is as follows.

Algorithm 2 Approximation algorithm for MaxFS

1: For each task i in task system T, let j_i^1 be the first job generated by task i; it has release date 0, deadline d_i, execution time c_i, and weight w_i, as defined in T. The cost b_i is defined as c_i/p_i. Let $\text{first}(T) = \{j_i^1 : i \in T\}$.
2: Apply the algorithm from Lemma 3.4 to the set $\text{first}(T)$, with budget $B = m$. Let $J' \subseteq \text{first}(T)$ be the feasible set of jobs returned by the algorithm.
3: Let T' be the set of tasks corresponding to J', that is, $T' = \{i : j_i^1 \in J'\}$. Output T'.

We prove the performance bound of the above algorithm using the following two lemmata. Let T^* be a subset of tasks that is optimal for MaxFS.

Lemma 3.6. *The set T' returned by Algorithm 2 has total weight at least $1/\rho_m$ times that of T^*.*

Proof. Let $J^* = \{j_i^1 : i \in T^*\}$. Since T^* is feasible, jobs in J^* can be completed by some preemptive schedule on m processors and their total cost $\sum_{j_i^1 \in J^*} c_i/p_i$ is at most m, since no set of tasks with utilization larger than m can be feasible on m processors. Hence, J^* is a feasible set of jobs. Then, by Lemma 3.4, the set J^* has total weight at most ρ_m times that of J'. Equivalently, T' has total weight at least $1/\rho_m$ times that of T^*. □

Hence, if T' can be scheduled on m processors with speed $(3 - 1/m)$, we have a $(3 - 1/m)$-speed ρ_m-approximate algorithm. To show this, we prove a more general lemma as follows.

Lemma 3.7. *Let T be a set of tasks that satisfies the following two properties.*

1. *The set of jobs* $\text{first}(T)$ *can be completed on-time by m speed-x processors.*
2. *The total utilization of T, that is, $\sum_{i \in T} c_i/p_i$, is at most $m \cdot y$.*

Then T is EDF*-schedulable on m speed-$(x + y + 1 - 1/m)$ processors.*

Proof. We start by proving that for any $t \in \mathbb{N}$,

$$\text{fd}_T([0,t]) \le \sum_{i \in T} \frac{c_i}{p_i} \cdot t + \sum_{j \in \text{first}(T)} \text{fd}_T(j,[0,t]). \tag{8}$$

Inequality (8) can be proven by considering different tasks separately. If $t < d_i$, we have

$$\text{fd}_T(i,[0,t]) = (c_i - (d_i - t))^+ = \text{fd}_T(j_i^1,[0,t]) \le \frac{c_i}{p_i} \cdot t + \text{fd}_T(j_i^1,[0,t]).$$

On the other hand, if $t \ge d_i$,

$$\text{fd}_T(i,[0,t]) \le \frac{c_i}{p_i} \cdot t + c_i = \frac{c_i}{p_i} \cdot t + \text{fd}_T(j_i^1,[0,t]).$$

Inequality (8) follows by summing over tasks.

Property 1 of the hypothesis ensures that all jobs in $\text{first}(T)$ can be completed on-time by m speed-x processors. Then, by Proposition 2.12, we obtain

$$\sum_{j \in \text{first}(T)} \text{fd}_T(j,[0,t]) \le mxt.$$

Property 2 states that the total utilization of T is at most my, that is, $\sum_{i \in T}(c_i/p_i) \cdot t \le myt$. Hence, using (8),

$$\begin{aligned}\text{fd}_T([0,t]) &\le \sum_{i \in T} \frac{c_i}{p_i} \cdot t + \sum_{j \in \text{first}(T)} \text{fd}_T(j,[0,t]) \\ &\le myt + mxt \\ &\le (m(\sigma - 1) + 1)t,\end{aligned}$$

with $\sigma = x + y + 1 - 1/m$, and by Lemma 2.14 T is EDF-schedulable on m speed-σ processors. □

Theorem 3.8. *Algorithm 2 is $(3 - 1/m)$-speed ρ_m-approximate for* MaxFS *for synchronous task systems on m processors.*

Proof. By Lemma 3.6, the total weight of T' is at least $1/\rho_m$ times that of T^*. Note that the corresponding set J' is feasible on m unit-speed processors and the total utilization of T' is at most m because of the budget constraint. By Lemma 3.7, T' is EDF-schedulable on m speed-$(3 - 1/m)$ processors and the theorem follows. □

4 Open Problems

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

1. Is there a pseudopolynomial time algorithm for the feasibility problem in synchronous multiprocessor systems (or even synchronous uniprocessor systems)?
2. Is there a polynomial time algorithm for the feasibility problem in arbitrary multiprocessor systems with a fixed number of task types?
3. Is there a constant-speed, constant-approximation algorithm for MaxFS in arbitrary multiprocessor systems?

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

Acknowledgements.

We thank two anonymous referees whose comments helped to improve the presentation of the paper.

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] N. Alon, U. Feige, A. Wigderson and D. Zuckerman. Derandomized graph products. *Computational Complexity*, 5:60–75, 1995.

[3] E. Amaldi, V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1–2):181–210, 1995.

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

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] O. Goldreich. *Foundations of cryptography. Vol. I: Basic techniques*. Cambridge University Press, 2001.

[15] D. R. Heath-Brown. The number of primes in a short interval. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 389:22-63, 1988.

[16] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.

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

[18] B. Kalyanasundaram and K. Pruhs. Eliminating Migration in Multi-Processor Scheduling. *Journal of Algorithms*, 38(1):2–24, 2001.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] 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.

[25] 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.

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

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

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

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

[30] K. Pruhs and G. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151–156, 2007.

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

[32] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

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