A Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines

Nathan Fisher and Sanjoy Baruah
Department of Computer Science
The University of North Carolina at Chapel Hill
{fishern, baruah}@cs.unc.edu

Abstract

Current feasibility tests for the static-priority scheduling on uniprocessors of periodic task systems run in pseudo-polynomial time. We present a fully polynomial-time approximation scheme (FPTAS) for feasibility analysis in static-priority systems with arbitrary relative deadlines. This test is an approximation with respect to the amount of a processor’s capacity that must be “sacrificed” for the test to become exact. We show that an arbitrary level of accuracy, $\epsilon$, may be chosen for the approximation scheme, and present a run-time bound that is polynomial in terms of $\epsilon$ and the number of tasks, $n$.

Keywords: Real-time scheduling; Uniprocessor systems; Static-priority systems; Feasibility analysis; Fully polynomial-time approximation schemes.

1 Introduction

An exact feasibility test for the preemptive uniprocessor scheduling of sets of periodic tasks, each with its deadline parameter equal to its period, was introduced by Lehoczky, Sha, and Ding [5]. The test determines whether a set of tasks is feasible using the rate monotonic algorithm [8]. The response time of each task is calculated, and checked against its deadline. Audsley et al. [2] developed a feasibility test for sets of tasks in which deadlines are less than periods, and which are scheduled using the deadline monotonic algorithm [7]. Lehoczky [6] provided a more general feasibility test for periodic task systems where the relation between deadlines and periods may be arbitrary.

In each of the aforementioned feasibility tests, the running time of the test is dependent on the values of the parameters of the tasks in the task system. Thus, these are pseudo-polynomial time tests. Albers and Slomka [1] present a fully polynomial-time approximation scheme (FPTAS) for feasibility of a sporadic task system using a dynamic-priority scheduling algorithm. The feasibility test accepts as input the specifications of a task system and a constant $\epsilon$, $0 < \epsilon < 1$, and is an approximation scheme in the following sense:

If the test returns “feasible”, then the task set is guaranteed to be feasible on the processor for which it had been specified. If the test returns “infeasible”, the task set is guaranteed to be infeasible on a slower processor, of computing capacity $(1 - \epsilon)$ times the computing capacity of the processor for which the task system had been specified.

In this paper, we extend the results of Albers and Slomka to the domain of static-priority scheduling with arbitrary relative deadlines. That is, we present an FPTAS for static-priority feasibility analysis that makes a performance guarantee similar to the one above: for any specified value of $\epsilon$, the FPTAS correctly identifies, in time polynomial in the number of tasks in the task system, all task systems that are static-priority feasible (with respect to a given priority assignment) on a processor that has $(1 - \epsilon)$ times the computing capacity of the processor for which the task system is specified. We have previously shown that such an FPTAS exists for static-priority systems when relative deadlines are bounded by periods [4].

Since many static-priority feasibility-analysis algorithms (in particular, those based upon iterative convergence of response-time equations) have been observed to converge extremely rapidly in practice, it may be argued that such an FPTAS is not particularly useful.

*Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).
However, the presence or otherwise of such an FPTAS is interesting from a theoretical perspective as part of the ongoing debate concerning the relative merits of static-priority and dynamic-priority scheduling: since an FPTAS was recently obtained for dynamic-priority uniprocessor feasibility-analysis, it is of interest to know whether static-priority feasibility-analysis could be approximated as efficiently as dynamic-priority analysis for arbitrary relative deadlines. The FPTAS presented in this paper answers this question in the affirmative.

The running time of current exact feasibility tests for static-priority task systems depends on the ratio between the largest and smallest period. Therefore, the complexity of current feasibility tests for task systems with widely-varying periods may prohibit their use in automatic synthesis tools. The running time of the approximation proposed in this paper is completely independent of tasks’ periods, and depends only on the number of tasks and the accuracy constant, $\epsilon$. Thus, the approximation offers a reduction in complexity for many task sets, and its predictable worst-case run-time guarantees a quick estimate for automatic synthesis tools exploring a real-time system design space.

The remainder of this paper is organized as follows. We formally define our task model in the next section. We briefly summarize (Section 3.1) how the request-bound function abstraction, which plays a crucial role in the various static-priority feasibility tests mentioned above [5, 2, 6], can be approximated by a function that is easily computed, and which satisfies the property that its value “closely” tracks the exact value of the request-bound function. We review the FPTAS and some results from [4] in Section 3.2. We give an approximate test for a task system with arbitrary relative deadlines in Section 4. We prove the correctness of the approximation test for arbitrary deadlines in Section 5. Finally, in Section 6, we formally state the main result of this paper, the existence of an FPTAS for feasibility in static-priority systems.

2 Task Model

We consider both periodic and sporadic task models. A task $\tau_i = (e_i, d_i, p_i)$ is characterized by a worst-case execution requirement $e_i$ and (relative) deadline $d_i$. In the periodic task model, the period $p_i$ represents the exact interval between the arrival of jobs of $\tau_i$. In the sporadic task model, $p_i$ represents the minimum interarrival separation between jobs of $\tau_i$. Each job has a worst-case execution requirement equal to $e_i$ and a deadline that occurs $d_i$ time-units after its arrival time. A task system $\tau$ is composed of tasks $\tau_1, \ldots, \tau_n$, where $n$ is the number of tasks in the system. A periodic task system is synchronous if the first job of every task is released at the same time.

A feasibility test is a necessary and sufficient set of conditions for determining whether a given task system will meet all of its deadlines. In the remainder of this paper, we study approximation schemes for feasibility of both sporadic and synchronous periodic task systems that are to be executed on a preemptive uniprocessor platform.

Static-Priority Scheduling Algorithms

In static-priority systems, each task is assigned a distinct priority, and all jobs of a task execute at the task’s priority. More formally, a job is said to be active at a specified time-instant in a schedule, if it has remaining execution time and has not missed its deadline. When a scheduling algorithm is invoked at time $t$, it will select the job with highest priority out of the set of active jobs at time $t$. Two well-studied static-priority scheduling algorithms are rate monotonic (RM) and deadline monotonic (DM). RM, introduced in [8], assigns each task a priority equal to the inverse of its period. DM, first presented in [7], assigns each task a priority equal to the inverse of its relative deadline. We will assume throughout this paper that tasks are indexed according to their assigned priority (i.e. for $1 \leq i < n$, $\tau_i$ has higher priority than $\tau_{i+1}$).

A task system $\tau$ is feasible with respect to static-priority systems if there exists a task priority assignment such that when $\tau$ is scheduled according to this priority assignment, all deadlines are met. A static-priority scheduling algorithm $A$ is optimal over all static-priority algorithms, if for every feasible task system $\tau$, $A$ produces a schedule in which all deadlines are met. RM is known to be optimal for static-priority algorithms when deadlines are equal to periods. DM is optimal for static-priority algorithms when deadlines are less than or equal to periods. Shih et al. [9] give a modified rate monotonic algorithm that is optimal for special cases when deadlines can exceed periods; however, to the best of our knowledge, there is currently no known optimal static-priority scheduling algorithm for arbitrary relative deadlines.

3 Bounded Relative Deadlines

In this section, we present the approximate feasibility test for static-priority systems, developed in [4],
where each task’s relative deadline is constrained to be at most its period. We begin in Section 3.1 by defining a request-bound function (RBF) that bounds the amount of execution time requested by a task (similarly defined in [5, 2, 6]). An approximation to the RBF is defined such that the deviation from the RBF is bounded.

In Section 3.2, we define both exact and approximate cumulative request-bound functions based, respectively, on the exact and approximate request-bound functions for a task \( \tau \). The functions describe the cumulative execution requests over a time interval for task \( \tau \) and all tasks of higher priority. Lehoczky et al. [5] showed that in a sporadic or synchronous periodic system, the smallest fixed point of task \( \tau \)’s cumulative request-bound function is no larger than its relative deadline, then \( \tau \) will always meet its deadline. If the smallest fixed point exceeds \( \tau \)’s deadline, then we cannot guarantee \( \tau \) will meet all deadlines; hence, \( \tau \) is not feasible.

### 3.1 Request-Bound Function

In a periodic synchronous task system, the total execution time requested by a task \( \tau \) can be expressed as a function of time. Every time a task \( \tau \) releases a job, \( e_i \) additional units of processor time are requested. The following function provides an upper bound on the total execution time requested by task \( \tau \) over time interval \([0, t]\):

\[
\text{RBF}(\tau, t) \triangleq \left\lceil \frac{t}{p_i} \right\rceil e_i
\]

(1)

In a sporadic task system, \( \text{RBF}(\tau, t) \) represents the total execution time requested by \( \tau \) in its worst-case phasing. Figure 1 shows an example of a RBF. Notice that the “step” function, \( \text{RBF}(\tau, t) \) increases by \( e_i \) units every \( p_i \) time units.

#### Approximating the RBF

The function \( \text{RBF}(\tau, t) \) has a discontinuity every \( p_i \) time units. We call these discontinuities steps. We define an approximation that computes the first \((k-1)\) steps of \( \text{RBF}(\tau, t) \) exactly (where \( k \) is a constant, defined below), and is a linear approximation of \( \text{RBF}(\tau, t) \), thereafter.

We choose a constant \( k \) based on our given “accuracy” constant \( \epsilon \), \( 0 < \epsilon < 1 \). For the remainder of the paper, assume the integer constant \( k \) is defined as follows:

\[
k \triangleq \left\lceil \frac{1}{\epsilon} \right\rceil - 1
\]

(2)

We now define the following function \( \delta(\tau, t) \) which closely approximates the function \( \text{RBF}(\tau, t) \):

\[
\delta(\tau, t) = \begin{cases} 
\text{RBF}(\tau, t), & \text{for } t \leq (k-1)p_i \\
e_i + \frac{e_i}{p_i}t, & \text{for } t > (k-1)p_i 
\end{cases}
\]

(3)

Figure 1 shows that \( \delta(\tau, t) \) is exactly \( \text{RBF}(\tau, t) \) up to \( t = (k-1)p_i \) (in this example, \( k = 3 \)) and then is a linear approximation for \( t > (k-1)p_i \) that “bounds” \( \text{RBF}(\tau, t) \) from above.

### 3.2 Description of Feasibility Test

#### Exact Test

For static-priority task systems with relative deadlines bounded by periods, Liu and Layland [8] showed that the worst-case response time for a job of task \( \tau \) occurs when all tasks of priority greater than \( \tau \) release a job simultaneously with \( \tau \). If a task \( \tau \) releases a job \( J \) simultaneously with all higher priority tasks and each higher priority task \( \tau \) releases subsequent jobs at the earliest legal opportunity (i.e. the inter-arrival separation between jobs of higher-priority task \( \tau \) is exactly \( p_j \)), then \( J \) has the largest response time of any job of task \( \tau \). In a sporadic or synchronous periodic task system with relative deadlines bounded by periods, it is necessary and suffi-
cient to only check the response time of the first job of each task. If the response time of the first job of task $\tau_i$ is at most its relative deadline, then $\tau_i$ is schedulable; else, it is not schedulable. A task system $\tau$ is feasible on a uniprocessor if and only if the first job of each task $\tau_i$ has a worst-case response time at most $d_i$.

In order to determine the response-time for the first job of task $\tau_i$, we must consider execution requests of $\tau_i$ and all jobs of tasks which may preempt $\tau_i$. We define the following cumulative request-bound function based on $\text{RBF}$. Let $T_{\tau_i}$ be the set of tasks with priority greater than $\tau_i$. Then, the cumulative request-bound function is defined as:

$$W_{\tau_i}(t) \overset{\text{df}}{=} \ell e_i + \sum_{\tau_j \in T_{\tau_i}} \text{RBF}(\tau_j, t) \tag{4}$$

The cumulative request-bound function $W_{\tau_i}(t)$ is simply the total execution requests of all tasks of higher priority than $\tau_i$ over the interval $(0, t]$, and the execution request of the first $\ell$ jobs of $\tau_i$. When deadlines do not exceed periods, we are concerned only with $W_{\tau_i}(t)$ which is the cumulative request-bound function for the first job of $\tau_i$.

Audsley et al. [3] presented an exact feasibility test for task $\tau_i$ using $\text{OM}$: a task is feasible if and only if there exists a fixed point, $t$, of $W_{\tau_i}(t)$ such that $t$ occurs before $\tau_i$'s deadline. The following theorem restates their test:

**Theorem 1 (from [3])** In a synchronous periodic (or sporadic) task system, task $\tau_i$ is feasible using $\text{OM}$ if and only if $\exists t \in (0, d_i]$ such that $W_{\tau_i}(t) \leq t$.

Approximate Test

The goal of using a linear approximation in $\delta(\tau_i, t)$ is to bound the number of steps in the approximation function. Since $\delta(\tau_i, t)$ has at most $k - 1$ steps for all $\tau_i$, a superposition of $\delta(\tau_i, t)$'s (i.e. a summation of a number of different $\delta$ functions) will have a polynomially bounded number of steps in terms of $k$ and the number of functions in the superposition. The following equation defines a superposition which we will use as the approximate cumulative request-bound function for the approximate feasibility test:

$$\hat{W}_{\tau_i}(t) \overset{\text{df}}{=} \ell e_i + \sum_{\tau_j \in T_{\tau_i}} \delta(\tau_j, t) \tag{5}$$

The following lemmas describe the implications of using the approximation function $\delta$ in the request-bound function. Informally, the first Lemma 1 states that if the approximate cumulative request-bound function is below line $f(t) = t$, then the exact cumulative request-bound function must be below as well.

**Lemma 1 (from [4])** If $\hat{W}_{\tau_i}(t) \leq t$, then $W_{\tau_i}(t) \leq t$.

Lemma 2 states that if the approximate cumulative request-bound function lies above $f(t) = t$, then the exact cumulative request-bound function must lie above the line $f(t) = \frac{k}{T_\tau}$. Formally stated:

**Lemma 2 (from [4])** If $\hat{W}_{\tau_i}(t) > t$, then $W_{\tau_i}(t) > \frac{k}{T_\tau}(t)$.

The value of $\hat{W}_{\tau_i}(t)$ is always at least that of $W_{\tau_i}(t)$. Therefore, if we use the approximate request-bound function, $\hat{W}_{\tau_i}(t)$ to find a fixed-point as in Theorem 1, the test is no longer necessary and sufficient. Instead, we will have a sufficient test for feasibility tests, as the following theorem states:

**Theorem 2 (from [4])** A synchronous periodic (or sporadic) task system, task $\tau_i$ is feasible using $\text{OM}$ if $\exists t \in (0, d_i]$ such that $\hat{W}_{\tau_i}(t) \leq t$.

Using the approximate response time function also no longer gives an exact check for infeasibility. Instead, if we cannot find a $t \in (0, d_i]$ such that $\hat{W}_{\tau_i}(t) \leq t$, then $\tau_i$ is infeasible on a lower capacity processor. In fact, we can quantify a smaller capacity processor for which this approximate feasibility test would become exact. The following theorem quantifies this capacity:

**Theorem 3 (from [4])** If $\forall t \in (0, d_i], \hat{W}_{\tau_i}(t) > t$, then $\tau_i$ is infeasible using $\text{OM}$ on a processor of $(1 - \epsilon)$ capacity.

The preceding theorem states we must effectively ignore $(1 - \epsilon)$ of the processor capacity for the test to become exact.

Together, theorems 2 and 3 provide an approximate test for feasibility of task system $\tau$. The time complexity of the test is $O(n^2k)$.

4 Arbitrary Relative Deadlines

When deadlines can exceed periods, Lehoczky [6] shows that it is no longer sufficient to check the response-times of only the first job of each task. Instead, it is potentially necessary to check the response-time of
all jobs in the level-i busy interval for each task \( \tau_i \). A level-i busy interval is a time interval \([a, b]\) where only jobs of \( T_i = T_{\tau_i} \cup \{\tau_i\} \) are executing continuously and the following is true:

1. A job of \( T_i \) is released at time \( a \).
2. All jobs of \( T_i \) released prior to \( a \) have completed by time \( a \).
3. \( b \) is the first time instant such that all jobs of \( T_i \) released in the interval \([a, b]\) have completed.

It may be tempting to try extending our results to a task system with arbitrary deadlines by applying the approximate feasibility test presented in this paper to each job of \( \tau_i \) in the level-i busy interval. Unfortunately, if this approach is used the approximate feasibility test is no longer polynomial in terms of \( n \) and \( \epsilon \). Applying the approximate feasibility test to each job of \( \tau_i \) in the level-i busy interval results in a pseudo-polynomial time test. The reasons that the test is pseudo-polynomial are the following:

- The length of the level-i busy interval does not depend on \( n \), but on the \( p_i \) and \( e_i \) terms; therefore, the level-i busy interval contains a pseudo-polynomial number of jobs of \( \tau_i \). Applying the approximate feasibility test of the previous section would require running the test a pseudo-polynomial number of times.

- The number of jobs of a task \( \tau_i \) that are active at each time instant \( t \) (i.e., \( t \) lies between the job’s release time and absolute deadline) could be \( \Theta(d_i/p_i) \). Again, this is not polynomial in terms of \( n \) and \( \epsilon \). Therefore, at each point in the testing set, we may have to perform a computation for a pseudo-polynomial number of active jobs to check if any of them have missed their deadline.

In this section, we show that pseudo-polynomial time checks are not required. We construct an FPTAS for feasibility analysis in static-priority systems with arbitrary relative deadlines given an arbitrary priority assignment. Furthermore, our FPTAS for arbitrary relative deadlines achieves the same asymptotic time complexity as the FPTAS for bounded relative deadlines. We define an algorithm that determines (according to the approximation functions defined in Section 3.2) the set of jobs of \( \tau_i \) that complete prior to or at time \( t = \max_{j \in [1, \ldots, |\tau|]} (k - 1)p_j \) (the point after which the approximation becomes a linear function), and meet their deadlines, in Section 4.1.1.

Section 4.1.2 describes a test to approximate the set of jobs that complete after time \( \max_{j \in [1, \ldots, |\tau|]} (k - 1)p_j \) and meet their deadline. We provide a proof sketch for the correctness of the feasibility approximation in Section 5. We derive the running time of approximation in Section 5.1. For the lemmas and theorems of the following section, we provide a proof sketch and intuition. The extended version of this paper will include full proofs for each of these lemmas and theorems.

4.1 Feasibility Test

4.1.1 Jobs with completion time prior or equal to \( \max_{j \in [1, \ldots, |\tau|]} (k - 1)p_j \)

We define the following function used to determine the set of jobs that have their execution requests satisfied by time \( t \):

\[
\tilde{W}_i(t) = \delta_{\tau_i, t} + \sum_{\tau_j \in T_i} \delta_{\tau_j, t} \quad (6)
\]

\( \tilde{W}_i(t) \) represents a “close” approximate to the cumulative requests of task \( \tau_i \) and all higher priority tasks with respect to \( t \) given time \( t \). In comparison, \( \hat{W}_{i,t}(t) \) is only a “close” approximation when \( t \) lies in the interval \((t - 1)p_i, t] \). An example of \( \tilde{W}_i(t) \) and \( \hat{W}_{i,t}(t) \) functions is illustrated in Figure 2.
Notice that the number of active jobs at time $t$ could be $\Theta(d_i/p_i)$. The next function is used to identify the index of the most recently released job of $\tau_i$ to have its execution request satisfied by time $t$.

$$Z_i(t) = \max\left(\frac{\hat{W}_i(t) - t}{e_i}, 0\right)$$ (7)

The index of the most recently released job to have its execution request satisfied by time $t$ is $\left\lceil \frac{Z_i(t)}{p_i} \right\rceil - Z_i(t)$ (according to our approximation). By finding the index of the most recently released job of $\tau_i$, we can determine the set of jobs of $\tau_i$ that do not have their execution requests satisfied at or prior to $t$.

**Lemma 3** If $\left\lceil \frac{Z_i(t)}{p_i} \right\rceil - Z_i(t) \geq 1$ and $t \leq (k - 1)p_i$, then for $\ell \in \{1, \ldots, \left\lceil \frac{Z_i(t)}{p_i} \right\rceil\}$ the $\ell$th job has its request satisfied by $t$ (i.e., $W_i(t) \leq t$).

**Proof Sketch:** Let $b = \left\lceil \frac{Z_i(t)}{p_i} \right\rceil - Z_i(t)$. There are two cases:

1. $Z_i(t) > 0$
2. $Z_i(t) = 0$

In both cases, we can derive the inequality $\hat{W}_{i,b}(t) \leq t$, and the lemma follows. ■

The next lemma shows: if the $\ell$th job of $\tau_i$ is active at time $t$, and its index exceeds $\left\lceil \frac{Z_i(t)}{p_i} \right\rceil$, then the $\ell$th job does not complete before or at time $t$. Using this lemma we can determine the set of jobs of $\tau_i$ that do not have their execution request satisfied by time $t$.

**Lemma 4** If $\ell \geq \left\lceil \frac{Z_i(t)}{p_i} \right\rceil - Z_i(t)$ and $t > (\ell - 1)p_i$, then $\hat{W}_{i,b}(t) > t$.

**Proof Sketch:** Suppose that for job $a = \left\lceil \frac{Z_i(t)}{p_i} \right\rceil - Z_i(t) + 1$, $\hat{W}_{i,a}(t) \leq t$. Then, the difference between $\hat{W}_i(t)$ and $t$ is less than $\hat{W}_i(t) - \hat{W}_{i,a-1}(t)$ (since $t \geq \hat{W}_{i,a-1}(t) > \hat{W}_{i,a}(t)$). We can show $Z_i(t)$ can be expressed by both Equation (7) and $\hat{W}_i(t) - \hat{W}_{i,a-1}(t)$. This will imply the following equality $\hat{W}_i(t) - \hat{W}_{i,a-1}(t) = \hat{W}_i(t) - t$; however, this contradicts the fact that $t > \hat{W}_{i,a-1}(t)$ is a strict inequality. ■

We now define, for a given task $\tau_i$, the set of points that must be tested in our approximation as:

$$\hat{S}_i \equiv \{ t = bp_a : a = 1, \ldots, \ell ; b = 1, \ldots, k - 1 \} \cup \{0\}$$ (8)

We call two elements $t_1$ and $t_2$ ($t_1 < t_2$) in set $\hat{S}_i$ adjacent if no $t$ satisfying $t_1 < t < t_2$ is in $\hat{S}_i$. Observe that for any two adjacent elements $t_1, t_2 \in \hat{S}_i$, $|t_2 - t_1| \leq p_i$. Therefore, at most one job of $\tau_i$ can have its deadline occur between any two adjacent elements of $\hat{S}_i$.

**ApproxFirstStage($\tau, i, k$):**

**Step 0:** Construct an ordered set $\hat{S}_i$ as in Equation (8).

**Step 1:** Initialize variable set $\hat{S}_i$, as in Equation (8).

**Step 2:** For each $t_i \in \hat{S}_i$, $(\{0\})$:

a) If $t_i > (\text{lowest active} - 1)p_i + d_i$ then:

   i) Let $t_{a-1}$ be the adjacent element prior to $t_i$ in ordered set $\hat{S}_i$. Let $\gamma$ be the total execution requirement of all jobs released at time $t_{a-1}$. Determine where the line defined by $(t_{a-1}, \hat{W}_i(\text{lowest active}(t_{a-1}) + \gamma))$ and $(t_i, \hat{W}_i(\text{lowest active}(t_i)))$ intersects with $f(t) = t$. Let this point of intersection be $t'$. (Note: $t'$ may not be in set $\hat{S}_i$.)

ii) If $t' > (\text{lowest active} - 1)p_i + d_i$, then return “$\tau_i$ is not schedulable”; otherwise, increment lowest active.

b) Let $x := \max (\{0\} - Z(t))$.

c) Let $\text{lowest active} := \max (x + 1, \text{lowest active})$.

**Step 3:** Return lowest active.

**Figure 3.** The function $\text{ApproxFirstStage}$ determines whether all jobs of task $\tau_i$ with deadlines less than $\max_{j \in \{1, \ldots, \ell\}}((k - 1)p_j)$ are schedulable. If no deadlines are missed up to time $\max_{j \in \{1, \ldots, \ell\}}((k - 1)p_j)$ $\text{ApproxFirstStage}$ returns the lowest indexed job whose execution request is not satisfied by time $\max_{j \in \{1, \ldots, \ell\}}((k - 1)p_j)$. Otherwise, it returns $\tau_i$ not schedulable.
4.1.2 Jobs with completion time after
\[ \max_{j \in \{1, \ldots, \ell\}} ((k-1)p_j) \]
Next, we describe a constant time test for the set of jobs of task \( \tau \), that have deadlines after time \( \max_{j \in \{1, \ldots, \ell\}} ((k-1)p_j) \) and \textit{ApproxFirstStage} does not determine that their demand is satisfied prior to or at time \( \max_{j \in \{1, \ldots, \ell\}} ((k-1)p_j) \). Notice from the definition of \( \hat{W}_{\ell,i} \),
\[
\forall \ell \in \mathbb{N}, \forall t \in (\max_{j \in \{1, \ldots, \ell\}} ((k-1)p_j), \infty) \colon (\hat{W}_{\ell,i}(t) = t e_i + \sum_{j \in T_i} (e_j + \frac{\Delta}{p_j}))) \tag{9}
\]
Let us assume that \textit{ApproxFirstStage}(\( \tau, i, k \)) returns \( h \). This means that \( h \) is the lowest indexed job of \( \tau \) that according to the approximation has not had its execution request satisfied by time \( \max_{j \in \{1, \ldots, \ell\}} ((k-1)p_j) \). Then for all \( \ell \in \mathbb{N}(\ell \geq h) \), we can solve equation (9) to find point \( t_{\ell} = \hat{W}_{\ell,i}(t_{\ell}) = t_{\ell} \).
\[
t_{\ell} = \frac{\ell e_i}{1 - U_{-i}} + \sum_{j \in T_i} \frac{e_j}{1 - U_{-j}} \tag{10}
\]
where \( U_{-j} = \sum_{i \in T_i \setminus j} e_i \). From Lemma 1, we know that \( W_{\ell,i}(t_{\ell}) \geq t_{\ell} \). Intuitively, \( t_{\ell} \) represents the time at which the approximation determines that the processor can satisfy the execution requests of the \( \ell \)th job of task \( \tau_i \). Therefore, if
\[
t_{\ell} \leq (h-1)p_i + d_i, \tag{11}
\]
then the \( h \)th job of \( \tau_i \) meets its deadline. Otherwise, we declare \( \tau_i \) to be \textit{not schedulable}.

If Inequality (11) is true, we must then determine whether all subsequent jobs of \( \tau_i \) after \( h \) meet their deadlines. This is equivalent to determining whether \( \forall \ell \in \mathbb{N}(\ell > h), t_{\ell} \leq (\ell - 1)p_i + d_i \). Define
\[
\Delta_{\ell} \equiv [(\ell - 1)p_i + d_i] - \left[ \frac{\ell e_i}{1 - U_{-i}} + \sum_{j \in T_i} \frac{e_j}{1 - U_{-j}} \right]. \tag{12}
\]
\( \Delta_{\ell} \) represents the difference between \( t_{\ell} \) and the deadline for the \( \ell \)th job of task \( \tau_i \). The following lemma quantifies this difference in terms of \( \Delta_{\ell} \).

**Lemma 5** \( \forall \ell(\in \mathbb{N}) \geq h, \Delta_{\ell} = \Delta_h - (\ell - h)\left( \frac{\epsilon_{i}}{1 - U_{-i}} - p_i \right) \).

**Proof Sketch:** The proof is easily shown by induction on \( \ell \).

Using the previous lemma, we can show that if the \( h \)th job of task \( \tau_i \) meets its deadline, then all subsequent jobs of \( \tau_i \) will meet their deadlines \textit{if and only if} \( \frac{\epsilon_{i}}{1 - U_{-i}} > p_i \). The next lemma formalizes this statement.

**Lemma 6** Given that \( \Delta_h \geq 0 \), then \( \exists \ell \in \mathbb{N}(\ell > h) \) such that \( t_{\ell} > (\ell - 1)p_i + d_i \) if and only if \( \frac{\epsilon_{i}}{1 - U_{-i}} > p_i \).

**Proof:** We will prove the \textit{only if} part, first. Assume that \( t_{\ell} > (\ell - 1)p_i + d_i \) and \( \ell > h \). Notice from equation (10),
\[
t_{\ell} - t_h = (\ell - h)\frac{\epsilon_{i}}{1 - U_{-i}}. \tag{11}
\]
By equation (11),
\[
t_{\ell} - t_h \geq (\ell - 1)p_i + d_i - (h-1)p_i - d_i \tag{11}
\]
which implies \( \frac{\epsilon_{i}}{1 - U_{-i}} > p_i \).

Now proving the \textit{if} direction, assume that \( \frac{\epsilon_{i}}{1 - U_{-i}} > p_i \). Define \( \ell \) as follows:
\[
\ell = \left[ \frac{\Delta_h}{\left( \frac{\epsilon_{i}}{1 - U_{-i}} - p_i \right)} \right] + 1 + h
\]
Obviously, \( \ell > h \). We will show that for the \( \ell \)th job of task \( \tau_i \), \( t_{\ell} > (\ell - 1)p_i + d_i \). Define
\[
\Delta_{\ell} = \Delta_h - (\ell - h)\left( \frac{\epsilon_{i}}{1 - U_{-i}} - p_i \right) \tag{from Lemma 5}
\]
\[
\leq \Delta_h - \left( \frac{\Delta_h}{\left( \frac{\epsilon_{i}}{1 - U_{-i}} - p_i \right)} \right) + 1 \tag{from definition of \( \ell \)}
\]
\[
\leq \Delta_h - \Delta_h - \left( \frac{\epsilon_{i}}{1 - U_{-i}} - p_i \right) \tag{from assumption}
\]
\[
\leq 0 \tag{from assumption}
\]
\( \Delta_{\ell} < 0 \) implies \((\ell - 1)p_i + d_i - t_{\ell} < 0 \). Thus, \( t_{\ell} > (\ell - 1)p_i + d_i \).

We have shown that we can check the approximate feasibility of the \( h \)th job and all subsequent jobs of task \( \tau_i \) by testing inequality (11) and checking that \( \frac{\epsilon_{i}}{1 - U_{-i}} \leq p_i \). Figure 4 gives the pseudo-code for the algorithm \textit{ApproxSecondStage}. Finally, Figure 5 describes the full approximation scheme for feasibility of synchronous periodic or sporadic static-priority task systems with respect to a given priority assignment.

5 Proof of Correctness for Arbitrary Deadlines

In this section, we will give a proof sketch of correctness for \textit{Approx}. The goal is to show that:

If \textit{Approx}(\( \tau, e \)) returns “feasible”, then the task
Lemma 7 After \( a - 1 \) iterations and prior to the \( a \)th iteration of the for loop of ApproxFirstStage, the following condition holds:

\[
\forall a \in \{1, \ldots, \text{lowest_active}_a - 1\} :: \\
(\exists \ell \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i) : \bar{W}_{1,\ell}(t) \leq t)
\]

Proof Sketch: We can show the invariant by induction. Observe that the base case is vacuously true. Notice that at each step of the algorithm, the variable lowest_active can either increase or remain the same as the previous iteration. If the value of lowest_active remains the same, then by the inductive hypothesis the invariant still holds. However, if the value of lowest_active increases, the invariant holds due to Lemma 3.

In [6], Lehoczky showed: if for each job \( j \) of task \( \tau_i \) there exists a time \( t \) between the release and deadline of \( j \) such \( W_{i,j}(t) \leq t \), then \( \tau_i \) is schedulable. We will use this result to show that the task set \( \tau \) is feasible when \( \text{Approx}(\tau, \epsilon) \) returns “\( \tau \) is feasible,” and \( \tau \) is infeasible on a processor of \( (1 - \epsilon) \) capacity when \( \text{Approx}(\tau, \epsilon) \) returns “\( \tau \) is infeasible.” We restate Lehoczky’s results in the following theorem.

Theorem 4 (from [6]) A sporadic or synchronous periodic task system \( \tau \) is feasible if and only if for each task \( \tau_j \) there exists a time \( t \) between the release and deadline of \( \tau_j \) such \( W_{i,j}(t) \leq t \).

Proof: Before proving that Approx correctly identifies the feasible tasks, we will restate the following result from [4] used in the proof of Lemma 9:

Lemma 8 (from [4]). For adjacent elements, \( t_1, t_2 \in \bar{S}_i \), if \( \bar{W}(t_1) > t_1 \) and \( \bar{W}(t_2) > t_2 \), then \( \bar{W}(t) > t, \forall t \) in
Lemma 9 \text{Approx}(\tau, \epsilon) returns “\tau is feasible” if and only if

\[
\forall \tau_i \in \tau, \ell \in \mathbb{N}(\ell > 0) ::
(\exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] : \hat{W}_{i,\ell}(t) \leq t).
\]  

(13)

Proof Sketch: Proving the “only if” direction first, assume that \text{Approx}(\tau, \epsilon) returns “\tau is feasible.” We can show that for all \tau_i, Lemma 7 implies that for all jobs \ell with deadline prior to \text{max}_{\ell=1,...,d_i}([k - 1) p_i] there exists a \ell \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] such that \hat{W}_{i,\ell}(t) \leq t. Lemma 6 implies that if \text{ApproxSecondStage} returns “\tau_i is schedulable,” then all jobs \ell with deadlines after \text{max}_{\ell=1,...,d_i}([k - 1) p_i] there exists \ell \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] such that \hat{W}_{i,\ell}(t) \leq t. Together these existential statements imply Equation 13.

For the “if” direction, we can prove the contrapositive. In other words, we will assume that \text{Approx}(\tau, \epsilon) returns “\tau is infeasible.” It must be the case that some job \tau_i of task system \tau has been declared “not schedulable” by either \text{ApproxFirstStage} or \text{ApproxSecondStage}. There are two cases:

1. \text{ApproxFirstStage} returns “\tau_i is not schedulable.”
2. \text{ApproxSecondStage} returns “\tau_i is not schedulable.”

In either case, we can find a job \ell of \tau_i, such that for all \ell \in ((a - 1)p_i, (a - 1)p_i + d_i], \hat{W}_{i,\ell}(t) > t. Therefore, we have shown the negation of Equation 13.

The following theorem proves formally that if \text{Approx} declares “\tau is feasible,” then \tau is, in fact, feasible.

Theorem 5 A sporadic or synchronous periodic task system, \tau, is feasible if \text{Approx}(\tau, \epsilon) returns “\tau is feasible” (where 0 < \epsilon < 1).

Proof: If \text{Approx}(\tau, \epsilon) returns “\tau is feasible,” then by Lemma 9,

\[
\forall \tau_i \in \tau, \ell \in \mathbb{N} > 0, 
\exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] : \hat{W}_{i,\ell}(t) \leq t.
\]

By Lemma 1,

\[
\forall \tau_i \in \tau, \ell \in \mathbb{N} > 0, 
\exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] : \hat{W}_{i,\ell}(t) \leq t.
\]

The theorem follows by applying Theorem 4.

In the next theorem, we state the implications of \text{Approx}(\tau, \epsilon) returning “\tau is infeasible”:

Theorem 6 If for a sporadic or synchronous periodic task system, \tau, and \epsilon \in (0, 1), \text{Approx}(\tau, \epsilon) returns “\tau is infeasible,” then \tau is infeasible on a processor of capacity (1 - \epsilon).

Proof: The proof is by contradiction. Assume that \text{Approx}(\tau, \epsilon) returns “\tau is infeasible,” and \tau is feasible on a processor of capacity (1 - \epsilon). By Lemma 9,

\[
\forall \tau_i \in \tau, \ell \in \mathbb{N} ::
\forall t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i) : \hat{W}_{i,\ell}(t) > t.
\]

This implies from Lemma 2 that

\[
\exists \tau_i \in \tau, \ell \in \mathbb{N} ::
\forall t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i) : \hat{W}_{i,\ell}(t) > \frac{k}{\epsilon} t \geq (1 - \epsilon)t.
\]

However, if \tau_i is feasible on a processor of (1 - \epsilon) capacity, Theorem 4 implies \forall \tau_i \in \tau, \ell \in \mathbb{N}, \exists t \in ((\ell - 1)p_i, (\ell - 1)p_i + d_i] such that \hat{W}_{i,\ell}(t) \leq (1 - \epsilon)t. This is a contradiction; thus, the theorem is true.

Thus, by Theorems 5 and 6, \text{Approx} is correct.

5.1 Computational Complexity

The computational complexity of \text{Approx}(\tau, \epsilon) depends entirely on the size of the testing set, \tilde{S}_i. It is easy to see that the size of \tilde{S}_i is at most:

\[
1 + i(k - 1)
\]

This corresponds to the number of iterations that \text{ApproxFirstStage} must make for each task, \tau_i. In our implementation of approximate feasibility-analysis, the condition in Step 2 of \text{ApproxFirstStage} would be executed at most \sum_{i=1}^{\tilde{m}}(1 + i(k - 1)) times, which is \mathcal{O}(n^2k).
6 Fully Polynomial-Time Approximation Scheme

For a given accuracy, $\epsilon$, the running time of the approximation algorithm is $O(n^2/\epsilon)$. Thus, these algorithms are members of a family of algorithms that collectively represent a fully polynomial-time approximation scheme for uniprocessor feasibility analysis, with respect to a given priority assignment, for both synchronous period and sporadic tasks systems in a static-priority system. The following theorem states this formally.

**Theorem 7** For any $\epsilon$ in the range $(0, 1)$, there is an algorithm $A_\epsilon$ that has run-time $O(n^2/\epsilon)$ and exhibits the following behavior: On any synchronous periodic or sporadic task system $\tau$,

- if $\tau$ is infeasible on a unit-capacity processor then Algorithm $A_\epsilon$ correctly identifies it as being infeasible;
- if $\tau$ is feasible on a processor of computing capacity $(1 - \epsilon)$ then Algorithm $A_\epsilon$ correctly identifies it as being feasible;
- else Algorithm $A_\epsilon$ may identify $\tau$ as being either feasible or infeasible.

7 Summary

It has been shown [1] that there exists a fully polynomial-time approximation scheme (FPTAS) for uniprocessor feasibility analysis of sporadic task sets in dynamic-priority systems. We have constructed a similar FPTAS for static-priority feasibility analysis of uniprocessor synchronous periodic and sporadic task systems with arbitrary relative deadlines. We have, thus, shown that dynamic- and static-priority systems have equivalent approximate feasibility-analysis “tools” available.

The fully polynomial-time approximation tests presented in this paper offer a reduction in complexity for feasibility tests. These approximate feasibility tests may be useful for quick estimates of task system feasibility in automatic system-synthesis tools.

References