

# Approximate Schedulability Analysis

Samarjit Chakraborty      Simon Künzli      Lothar Thiele

Computer Engineering and Networks Laboratory

Swiss Federal Institute of Technology (ETH) Zürich

E-mail: {samarjit, kuenzli, thiele}@tik.ee.ethz.ch

## Abstract

*The schedulability analysis problem for many realistic task models is intractable. Therefore known algorithms either have exponential complexity or at best can be solved in pseudo-polynomial time, thereby restricting the application of the concerned models to a large extent. We introduce the notion of “approximate schedulability analysis” and show that if a small amount of “error” (which is specified as an input to the algorithm) can be tolerated in the decisions made by the algorithm, then this problem can be solved in polynomial time. Our algorithms are analogous to fully polynomial time approximation schemes in the context of optimization problems. We show that this concept of approximate schedulability analysis is fairly general and can be applied to any task model which satisfies certain “task-independence” assumptions. Lastly, we substantiate our theoretical results with experimental evidence and clearly show the tradeoffs between the running time of the schedulability analysis and the error incurred for various values of the input error parameter.*

## 1 Introduction

Schedulability analysis for any real-time task model is concerned with determining whether it is possible to assign to each job a processor time equal to its execution requirement, between its ready-time and its deadline. Unfortunately, for most realistic task models this problem is intractable (usually co-NP-hard) and therefore known algorithms either have exponential complexity, or at best can be solved in pseudo-polynomial time. Under these circumstances, current research in the real-time systems area has focussed either on obtaining efficient pseudo-polynomial algorithms for these models, or on finding polynomial time algorithms for special cases of these models. Either of these solutions, nevertheless, largely restricts the use of such models for the design and analysis of realistic systems.

The work presented in this paper remedies this situation to a large extent. It is based on the observation that if a small amount of error in the decisions made by a schedulability analysis algorithm is acceptable, then it is possible to design such algorithms to run in polynomial time. This idea is similar in spirit to obtaining approximation algorithms for NP-hard optimization problems [7]. Algorithms for *approximate schedulability analysis* are of the following form.

If a task set is schedulable then the algorithm is guaranteed to return the correct answer SCHEDULABLE. But if a task set is not schedulable then for some cases the algorithm might incorrectly return SCHEDULABLE as well. However, in such cases, it is guaranteed that no job can miss its deadline by a time interval which is “too large”. The maximum length of time by which a job can miss its deadline (in case the algorithm incorrectly returns SCHEDULABLE) is bounded and is parameterized by an input error parameter  $\varepsilon$ . The smaller the value of  $\varepsilon$ , the smaller is this time interval and the higher is the running time of the algorithm. Therefore,  $\varepsilon$  represents a tradeoff between the maximum error incurred and the running time of the algorithm.

In contrast to such *optimistic algorithms*, it is also possible to design *pessimistic algorithms* which always return the correct answer if a task set is not schedulable. However, if a task set is schedulable then the algorithm might err and incorrectly return NOT SCHEDULABLE. As in the previous case, the error incurred by such wrong decisions is bounded and is parameterized by  $\varepsilon$ . This means that task sets for which the algorithm incorrectly returns NOT SCHEDULABLE can load the processor “heavily”, in the sense that there exists time intervals over which the processor might be almost always occupied if the deadline of all jobs are to be met. Here, the length of time, within such time intervals, for which the processor can be idle is a measure of the error incurred. As before, the smaller the value of  $\varepsilon$ , the smaller is the error, but at the expense of increasing the running time.

For certain task models we additionally give a third class of algorithms which can incur a double-sided error, meaning that both SCHEDULABLE and NOT SCHEDULABLE answers can be wrong. However, we show that for such algorithms the maximum error in either direction is less than the error incurred for the equivalent optimistic and pessimistic algorithms.

The work in this paper leads to polynomial time algorithms for approximate schedulability analysis for various well known task models such as the Sporadic model due to Mok [8], the Multiframe model of Mok and Chen [9], the Generalized Multiframe model of Baruah *et al.* [3], and the recently proposed Recurring Real-Time task model of Baruah [2, 1]. The known algorithms for (exact) schedu-

ability analysis for all of these models are either pseudo-polynomial or have exponential running time. Besides this, the other rationale behind approximate schedulability analysis is that in many application domains such as embedded systems, it is difficult to evaluate the worst-case execution times of jobs accurately. This is due to factors such as caching and pipelining in embedded processors. In such cases, either the worst-case execution times of jobs are over-estimated, or it is acceptable for jobs to miss their deadlines by small amounts of time. In either case, an approximate schedulability analysis, in the sense we described above, would suffice for all practical purposes. In other domains such as multimedia applications or networking, although the execution requirements of jobs can be accurately determined, if a job misses its deadline by a small amount of time then the performance of the system (quality of audio or video) does not deteriorate significantly.

**Our contributions and relation to previous work.** To the best of our knowledge, the concept of approximate schedulability analysis was not investigated until very recently. Most of the research on obtaining efficient algorithms for schedulability analysis for different real-time task models focussed on designing either efficient pseudo-polynomial algorithms or polynomial time solutions for restricted versions of the models.

In [6] it was shown that for a certain task model consisting of a collection of “one-shot” task graphs, where the control flows from the *source* to the *sink* vertex of a task graph only once, the schedulability analysis problem is intractable. Further, a polynomial time algorithm for approximate schedulability analysis was given for such a collection of task graphs. However, no recurring behavior of these task graphs was considered, as done in the case of all real-time task models. In [5] the conditions for the non-preemptive version of the schedulability analysis of the same model was derived and it was shown that similar techniques for approximate schedulability analysis can be applied here as well.

The results of [6] and [5] do not extend to the case of recurring executions of the task graphs, which would then be equivalent to the Recurring Real-Time task model proposed by Baruah [2, 1]. The techniques in [6] and [5] also do not apply to other task models such as the Multiframed model or the Generalized Multiframed model. The work presented in this paper builds on the techniques given in [6] and shows that it is possible to derive polynomial time algorithms for approximate schedulability analysis for a variety of task models which satisfy certain *task-independence assumptions* given in [3]. These assumptions are extremely general and are satisfied by many task systems encountered in practice. As a consequence, it is possible to derive polynomial time algorithms for approximate schedulability analysis for models such as the Recurring Real-Time task

model, the Generalized Multiframed model and the Multiframed model.

In the next section we describe an abstract model of task systems which satisfies the task-independence assumptions. Following this, we describe our general framework for approximate schedulability analysis applied to this abstract task model. In Section 3 we describe the Recurring Real-Time task model which belongs to the described abstract framework. Section 4 describes our algorithms for the Recurring Real-Time task model, following which we show our experimental results for this model in Section 5, illustrating the tradeoffs between running time and accuracy for various values of the input error parameter. Finally, in Section 6 we briefly outline how our algorithms extend to other task models such as the Sporadic model, Multiframed model and the Generalized Multiframed model.

## 2 An Abstract Model of Task Systems

In this section we present an abstract model of task systems and a generic framework for (exact) schedulability analysis for models satisfying the conditions imposed by this abstract model (see also [3]). Based on this framework, we then outline two basic building blocks which make up our algorithms for approximate schedulability analysis. Concrete examples of these two building blocks are given in Sections 4 and 6 where we derive algorithms for various known task models.

A *task* in this abstract model generates a (possibly infinite) sequence of *jobs*. Each job is characterized by a *ready-time*, an *execution requirement*, and a *deadline*. A task set consists of a collection of such tasks, all of which are to be executed on a single shared processor and jobs are preemptable. We consider only the preemptive uniprocessor case here, but in general there can be multiple shared processors and a job once scheduled for execution need not be preemptable. The generation of jobs by a task is constrained by a set of rules, which for example might be that there is a minimum separation in time between the generation of two consecutive jobs by a task. Jobs generated according to these constraints are said to be *legal*. The schedulability analysis of a given task set is concerned with determining whether it is possible to assign to each job a processor time equal to its execution requirement between its ready time and its deadline, for all possible legal job sequences generated by tasks of the task set.

The rules that govern the generation of jobs by a task can be stated in the form of the following two *task independence assumptions*. (i) The runtime behavior of a task is independent of any other tasks in the system. (ii) The constraints according to which legal job sequences are generated can be specified without any references to *absolute time*. Assumption (i) states that each task generates jobs independently of the jobs generated by other tasks in the system. Therefore, it is not permissible, for example, to require a task to gen-

erate a job in response to a job generated by another task. Assumption 2 states that all temporal specifications defining the rules according to which jobs are generated by a task can only be relative to the time at which the task begins execution, or can be relative to the ready-time of another job of the same task. Therefore, a constraint like the ready-times of two consecutive jobs of a task must be separated by at least  $p$  time units, conforms to this requirement. Lastly, the time at which a task begins execution (i.e. the first job is generated) is not *a priori* known. For example, a task can begin execution in response to some external event.

The task independence assumptions do not assume anything about the interactions between the jobs. Once a job is generated, it executes independently of any other job in the system, including those generated by the same task.

These assumptions are followed by a wide variety of task systems such as the Sporadic model, Multiframe model, Generalized Multiframe model, and the Recurring Real-Time task model.

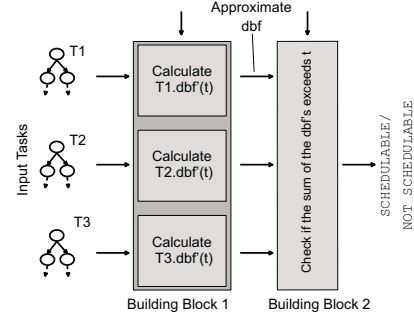
### 2.1 Exact Schedulability Analysis

For the abstract task model described above, a generic framework for schedulability analysis was presented in [3]. It is based on a generalization of the *processor demand criteria* introduced in [4]. In this framework, the worst case workload generated by a task is represented by a function called the *demand-bound function*. The demand-bound function of a task  $T$ , denoted by  $T.dbf(t)$ , takes as an argument a positive real number  $t$  and returns the maximum possible cumulative execution requirement by jobs that can be legally generated by  $T$  and which have their ready-times and deadlines within any time interval of length  $t$ . Based on this function, a method for schedulability analysis of a task system consisting of a set of tasks is given by the following theorem.

**Theorem 1 (Baruah et al. [3])** *A task set  $\mathcal{T}$  is schedulable if and only if for all  $t > 0$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) \leq t$ .*

Therefore,  $\mathcal{T}$  is not schedulable if and only if there exists some  $\hat{t}$  for which  $\sum_{T \in \mathcal{T}} T.dbf(\hat{t}) > \hat{t}$ . The schedulability analysis algorithms for many known task models (such as the Sporadic model, the Generalized Multiframe model, and the Recurring Real-Time task model) are based on identifying an upper bound  $t_{\max}$ , such that if  $\mathcal{T}$  is not schedulable then there exists some  $\hat{t} \leq t_{\max}$  for which  $\sum_{T \in \mathcal{T}} T.dbf(\hat{t}) > \hat{t}$ . Hence, a schedulability analysis algorithm is based on checking if  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is greater than  $t$  for all  $t \leq t_{\max}$ . If  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is less than or equal to  $t$  for all such values of  $t$  then the algorithm returns SCHEDULABLE, else it returns NOT SCHEDULABLE.

However, for all the models mentioned above,  $t_{\max}$  turns out to be pseudo-polynomial in the size of the input. Additionally, for some models such as the Recurring Real-Time task model, the problem of computing the value of  $T.dbf(t)$  for any  $t$  is NP-hard and also requires pseudo-polynomial



**Figure 1.** A framework for approximate schedulability analysis.  $\epsilon$   $\delta$  and  $\delta$  are the input error parameters.

time [6]. Hence, the overall algorithm for schedulability analysis for all these models run in pseudo-polynomial time. Since the schedulability analysis problem for some of these models (such as the Recurring Real-Time task model) is known to be intractable [6], at least for these models this is the best one could hope for in terms of asymptotic complexity (unless  $P = NP$ ). On the other hand, for models such as the Sporadic task model, the complexity of schedulability analysis is still not known.

### 2.2 A Framework for Approximate Schedulability Analysis

In its most general form, our framework for approximate schedulability analysis relies on the following two building blocks (see Figure 1). (i) Obtaining an approximation algorithm to compute the demand-bound function  $T.dbf(t)$  for any task graph  $T$  and time interval of length  $t$  in polynomial time. (ii) Instead of checking the value of  $\sum_{T \in \mathcal{T}} T.dbf(t)$  for all  $t = 1, \dots, t_{\max}$  (which can be pseudo-polynomial number of checks), only a polynomial number of checks are done.

Both the above two steps result in some *error*. The main contribution of this work is to show that if an appropriate polynomial time approximation algorithm exists for the first step then the *total error* incurred by the approximate schedulability analysis from the two steps is bounded (in a sense that we describe later) and it is possible to obtain a tradeoff between this error and the running time of the algorithm.

Let us assume that an approximation algorithm for computing the demand-bound function  $T.dbf(t)$  exists and it takes as an input an error parameter  $\epsilon$  and in polynomial time returns for any  $t$  an approximate value of the function denoted by  $T.dbf'(t)$ , such that  $T.dbf(t) \geq T.dbf'(t) \geq f(\epsilon)T.dbf(t)$ , where  $f$  is some function of  $\epsilon$ . Hence, we have

$$\frac{1}{f(\epsilon)}T.dbf'(t) \geq T.dbf(t) \geq T.dbf'(t) \quad (1)$$

For example, if the approximation algorithm is a fully-polynomial time approximation scheme (FPTAS) [7] then  $T.dbf \geq T.dbf'(t) \geq (1 - \epsilon)T.dbf(t)$  and hence the above inequality (1) takes the form  $\frac{1}{1-\epsilon}T.dbf'(t) \geq T.dbf(t) \geq T.dbf'(t)$ .

If the size of the input specification of the task  $T$  is  $O(n)$  and  $0 < \varepsilon < 1$ , then such an FPTAS for computing  $T.dbf'(t)$  for any  $t$  runs in  $poly(n, \frac{1}{\varepsilon})$  time (where  $poly$  denotes some polynomial function) and the smaller the value of  $\varepsilon$  the less is the error in estimating  $T.dbf(t)$ , however, at the cost of increasing the running time.

Hence, for any  $t \geq 0$ ,  $\frac{1}{f(\varepsilon)}T.dbf'(t)$  and  $T.dbf'(t)$  can be used as upper and lower bounds for  $T.dbf(t)$  and such bounds can be computed in polynomial time. Our approximate schedulability analysis is based on using either this upper or lower bound for  $T.dbf(t)$  to check if the sum of the demand-bound functions for all the tasks in the task set  $\mathcal{T}$  exceeds  $t$  for any value of  $t \leq t_{\max}$ .

Let us first suppose that we use the lower bound  $T.dbf'(t)$ , and for any  $t \leq t_{\max}$  return NOT SCHEDULABLE if  $\sum_{T \in \mathcal{T}} T.dbf'(t) > t$ . If  $\sum_{T \in \mathcal{T}} T.dbf'(t) \leq t$ , for all  $t \leq t_{\max}$  then we return SCHEDULABLE. Such an algorithm is overly *optimistic* in the sense that if a task set  $\mathcal{T}$  is schedulable then the algorithm is guaranteed to return the correct answer. However, for some task sets the algorithm might incorrectly return SCHEDULABLE even if they are not. In such cases, for some  $t \leq t_{\max}$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) > t$  but  $\sum_{T \in \mathcal{T}} T.dbf'(t) \leq t$ . Therefore, the error incurred is equal to  $\sum_{T \in \mathcal{T}} T.dbf(t) - \sum_{T \in \mathcal{T}} T.dbf'(t) \leq (1 - f(\varepsilon)) \sum_{T \in \mathcal{T}} T.dbf(t)$ .

Hence, for such a value of  $t$ , a job can miss its deadline by at most  $(1 - f(\varepsilon)) \sum_{T \in \mathcal{T}} T.dbf(t)$  time units (in the case of an FPTAS for approximating  $T.dbf(t)$ , this is equal to  $\varepsilon \sum_{T \in \mathcal{T}} T.dbf(t)$ ). For small values of  $\varepsilon$ , such an algorithm can therefore make an error only for task sets in which jobs can miss their deadlines only by small intervals of time.

Alternatively, it is possible to design a *pessimistic algorithm*, which for a task set  $\mathcal{T}$  returns SCHEDULABLE if  $\sum_{T \in \mathcal{T}} \frac{1}{f(\varepsilon)}T.dbf'(t) \leq t$ , for all  $t \leq t_{\max}$ , else it returns NOT SCHEDULABLE. Since  $\frac{1}{f(\varepsilon)}T.dbf'(t)$  is an overestimate of  $T.dbf(t)$ , such an algorithm always returns the correct answer if a task set is not schedulable. However, for certain task sets which are schedulable, this algorithm might return an incorrect answer. In such cases, for some  $t \leq t_{\max}$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) \leq t$  but  $\sum_{T \in \mathcal{T}} \frac{1}{f(\varepsilon)}T.dbf'(t) > t$ . The error incurred is equal to  $\sum_{T \in \mathcal{T}} \frac{1}{f(\varepsilon)}T.dbf'(t) - \sum_{T \in \mathcal{T}} T.dbf(t) \leq \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf(t)$ .

Therefore, task sets for which this algorithm can err are those which over some time interval of length  $t$  can load the processor for at least  $t - \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf(t)$  time units (which in the case of the FPTAS for approximating  $T.dbf(t)$  is  $t - \frac{\varepsilon}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf(t)$ ). Hence, for small values of  $\varepsilon$  these are task sets which in a sense can “heavily” load the processor.

The above algorithms by themselves do not result in a polynomial time algorithm for approximate schedulabil-

ity analysis if  $t_{\max}$  is pseudo-polynomial in the size of the problem specification, since then a pseudo-polynomial number of checks have to be done. To avoid this, if there are  $m$  tasks in  $\mathcal{T}$  then we instead perform a check only for  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$ , where  $K = \frac{\delta t_{\max}}{poly(m)}$ . Here  $\delta$  is an input error parameter to the algorithm (similar to  $\varepsilon$  in the case of approximating the demand-bound function) and  $poly(m)$  is any polynomial function of  $m$ . Hence, the total number of checks is now  $O(\frac{poly(m)}{\delta})$ , which is polynomial in the size of the input.

We now bound the error incurred by such an algorithm. Consider the algorithm where we check the value of  $\sum_{T \in \mathcal{T}} T.dbf(t)$  at  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$  and return NOT SCHEDULABLE if for any of these values of  $t$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) > t$  and else we return SCHEDULABLE. If  $\mathcal{T}$  is schedulable, then clearly such an algorithm always returns the correct answer. But for some task sets which are not schedulable, the algorithm might incorrectly return SCHEDULABLE as well. However, in such cases a job from such a task set might miss its deadline by at most  $K$  time units.

The idea behind the algorithm is that since  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is a non-decreasing function of  $t$ , if its value at some  $t'$  exceeds  $t'$  by a large number then even with a polynomial number of checks it would be possible to come across some  $t$  close to  $t'$  at which the value of the function exceeds  $t$ . To see this, consider some time interval  $[iK, (i+1)K]$ . If  $\sum_{T \in \mathcal{T}} T.dbf(iK) \leq iK$  and  $\sum_{T \in \mathcal{T}} T.dbf((i+1)K) = (i+1)K$  (and hence the condition for schedulability is met at these two points), then in the worst case  $\sum_{T \in \mathcal{T}} T.dbf(iK + \Delta) = (i+1)K$ , where  $\Delta \rightarrow 0$ , and hence a job from  $\mathcal{T}$  can miss its deadline by at most  $K$  time units. This argument applies to all intervals, the end points of which are only checked by the algorithm.

As in the case of the previous approximate schedulability analysis based on approximating the demand-bound function, here, too, it is possible to design a pessimistic algorithm. For this we check the value of  $\sum_{T \in \mathcal{T}} T.dbf(t)$  at  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$  and return NOT SCHEDULABLE if for any of these values of  $t$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) > t - K$  and else we return SCHEDULABLE.

If  $\mathcal{T}$  is not schedulable then this algorithm always returns the correct answer. To see this, suppose that  $\mathcal{T}$  is not schedulable. Then there exists an interval  $[iK, (i+1)K]$  such that for some  $t \in [iK, (i+1)K]$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) > t$ . Since  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is a non-decreasing function,  $\sum_{T \in \mathcal{T}} T.dbf((i+1)K) > iK$  and hence our algorithm returns NOT SCHEDULABLE. For task sets  $\mathcal{T}$  which are schedulable, this algorithm can return an incorrect answer and clearly the error in these cases is bounded by  $K$ , i.e. for such  $\mathcal{T}$  there exist time intervals of length  $t$  over which the processor can be occupied for at least  $t - K$  time units.

Since  $K = \frac{\delta t_{\max}}{\text{poly}(m)}$ , a smaller value of  $\delta$  reduces the maximum error that can be incurred by both the optimistic and the pessimistic algorithms, at the cost of increasing the number of checks to be performed and hence the running time of the algorithm.

### 2.2.1 Bounding the Total Error

In this section we present algorithms obtained by combining the two steps described above (i.e. approximating the demand-bound function, and performing a polynomial instead of a pseudo-polynomial number of checks to verify whether the sum of the demand-bound functions for any  $t$  exceeds  $t$ ). As before, we present two classes of algorithms—optimistic and pessimistic—and give a bound on the maximum error that can be incurred in both the cases. Both of these algorithms perform an approximate schedulability analysis in polynomial time, and the total error incurred depends on the values of the input error parameters  $\varepsilon$  and  $\delta$ . The smaller these values, the less is the error but at the cost of the running time increasing appropriately.

**Optimistic Algorithms.** For algorithms of this class, we check the value of  $\sum_{T \in \mathcal{T}} T.dbf'(t)$  at  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$  and return NOT SCHEDULABLE if for any of these values of  $t$ ,  $\sum_{T \in \mathcal{T}} T.dbf'(t) > t$  and else return SCHEDULABLE. As before,  $T.dbf'(t)$  is an approximate value of  $T.dbf(t)$  such that  $T.dbf(t) \geq T.dbf'(t) \geq f(\varepsilon)T.dbf(t)$  for any  $t$ . Note that this algorithm takes as input two error parameters  $\varepsilon$  and  $\delta$ .

If  $\mathcal{T}$  is schedulable then  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is less than or equal to  $t$  for all values of  $t$  at which this sum is checked, and since  $T.dbf(t) \geq T.dbf'(t)$ , our algorithm is guaranteed to return SCHEDULABLE.

Now consider the case where  $\mathcal{T}$  is not schedulable. Here our algorithm can return an incorrect answer, and there are two sources of possible error. Consider any  $t$  at which the value of  $\sum_{T \in \mathcal{T}} T.dbf'(t)$  is checked by our algorithm. If for any such  $t$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t) > t$  but  $\sum_{T \in \mathcal{T}} T.dbf'(t) \leq t$  then we incur a maximum error of  $(1 - f(\varepsilon)) \sum_{T \in \mathcal{T}} T.dbf(t)$  which is less than or equal to  $\frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(t)$ .

Secondly, consider any interval  $[iK, (i+1)K]$  such that the value of  $\sum_{T \in \mathcal{T}} T.dbf'(t)$  is checked at  $t = iK$  and  $t = (i+1)K$  and for both these values of  $t$ , the sum is less than or equal to  $t$ . The worst case error incurred in such a case occurs when  $\sum_{T \in \mathcal{T}} T.dbf'(iK + \Delta) = (i+1)K$  where  $\Delta \rightarrow 0$ , and therefore this error is equal to  $K$ . Taking into account that  $\sum_{T \in \mathcal{T}} T.dbf(iK) \leq \frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK)$ , the total error incurred by the algorithm within this interval is equal to  $K + \frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK) - \sum_{T \in \mathcal{T}} T.dbf(iK) \leq K + \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK)$ .

Since for  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$ , the value of  $T.dbf'(t)$  is maximized at  $t = (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$ , the maxi-

mum possible total error incurred by the algorithm is equal to  $K + \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'((\lfloor \frac{t_{\max}}{K} \rfloor + 1)K)$ .

**Pessimistic Algorithms.** Here we check the value of  $\sum_{T \in \mathcal{T}} \frac{1}{f(\varepsilon)} T.dbf'(t)$  at  $t = K, 2K, \dots, (\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$  and return NOT SCHEDULABLE if for any of these values of  $t$ ,  $\frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(t) > t - K$ . Following the reasoning given in Section 2.2, this algorithm is guaranteed to return the correct answer if  $\mathcal{T}$  is not schedulable. But it might err if  $\mathcal{T}$  is schedulable. To bound the error in such cases, suppose that for some  $iK$ ,  $1 \leq i \leq (\lfloor \frac{t_{\max}}{K} \rfloor + 1)$ ,  $\frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK) > (i-1)K$  (and therefore our algorithm returns NOT SCHEDULABLE) but  $\sum_{T \in \mathcal{T}} T.dbf(iK) \leq iK$ . Hence, the error incurred at  $iK$  is equal to  $K + \frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK) - \sum_{T \in \mathcal{T}} T.dbf(iK) \leq K + \frac{1}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK) - \sum_{T \in \mathcal{T}} T.dbf(iK) = K + \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'(iK)$ . Hence, the maximum error incurred by this algorithm is also equal to  $K + \frac{1-f(\varepsilon)}{f(\varepsilon)} \sum_{T \in \mathcal{T}} T.dbf'((\lfloor \frac{t_{\max}}{K} \rfloor + 1)K)$ .

The running times of both the optimistic and the pessimistic algorithms are polynomial, assuming that there exists a polynomial time approximation algorithm for computing  $T.dbf(t)$  for any task  $T$  and time interval of length  $t$ . For example, if the later algorithm is an FPTAS, and the specification of any task is of the size  $O(n)$  and  $\mathcal{T}$  contains  $m$  such tasks, then the total running time of any of the algorithms for approximate schedulability analysis is  $O(\text{poly}(m, n, \frac{1}{\delta}, \frac{1}{\varepsilon}))$ .

## 3 Recurring Real-Time Task Model

In this section we give the first example of a task model satisfying the constraints imposed by the abstract model of task systems described in Section 2. This model, called the Recurring Real-Time task model was introduced by Baruah [2, 1]. It is especially suited for accurately modeling conditional real-time code with recurring behavior, i.e. where code blocks have conditional branches and run in an infinite loop (see [1, 6, 5]).

A recurring real-time task  $T$  is represented by a task graph which is a directed acyclic graph with a unique source (a vertex with no incoming edges) and a unique sink (a vertex with no outgoing edges) vertex. Associated with each vertex  $v$  of this graph is its execution requirement  $e(v)$ , and deadline  $d(v)$ . Whenever the vertex  $v$  is *triggered*, it generates a job which has to be executed for  $e(v)$  amount of time within  $d(v)$  time units from the triggering-time. Each directed edge  $(u, v)$  in the graph is associated with a minimum intertriggering separation  $p(u, v)$ , denoting the minimum amount of time that must elapse before the vertex  $v$  can be triggered after the triggering of the vertex  $u$ .

The semantics of the execution of such a task graph state that the source vertex can be triggered at any time, and if some vertex  $u$  is triggered then the next vertex  $v$  can be trig-

gered only if there exists a directed edge  $(u, v)$  and at least  $p(u, v)$  amount of time has passed since the triggering of the vertex  $u$ . If there are directed edges  $(u, v_1)$  and  $(u, v_2)$  from the vertex  $u$  (representing a conditional branch) then only one among  $v_1$  and  $v_2$  can be triggered, after the triggering of  $u$ . The triggering of the sink vertex can be followed by the source vertex getting triggered again but any two consecutive triggerings of the source vertex should be separated by at least  $P(T)$  units of time, called the *period* of the task graph.

Therefore, a sequence of vertices  $v_1, v_2, \dots, v_k$  getting triggered at time instants  $t_1, t_2, \dots, t_k$ , is legal if and only if there are directed edges  $(v_i, v_{i+1})$ , and  $t_{i+1} - t_i \geq p(v_i, v_{i+1})$  for  $i = 1, \dots, k - 1$ . The only exception is that  $v_{i+1}$  can also be the source and  $v_i$  the sink vertex, and in that case if there exists some vertex  $v_j, j < i$ , in the sequence such that  $v_j$  is also the source vertex then  $t_{i+1} - t_j \geq P(T)$  must be additionally satisfied. The real-time constraints require that the job generated by triggering vertex  $v_i, i = 1, \dots, k$ , be assigned the processor for  $e(v_i)$  amount of time within the time interval  $(t_i, t_i + d(v_i)]$ .

Recall from Section 2 that the task independence assumptions require that once jobs are generated, they execute independently of each other (and therefore a restriction like first-come-first-served can not hold). Therefore, to ascertain that a job generated by a vertex  $u$  completes execution before a job generated by a vertex  $v$ , if there is a directed edge from  $u$  to  $v$ , either of the following conditions must hold:  $p(u, v) \geq d(u)$ , which guarantees that the vertex  $v$  can be triggered only after the job generated by vertex  $u$  has completed execution, or that  $d(u) \leq p(u, v) + d(v)$ , which guarantees that the absolute deadline of the job generated by vertex  $v$  is larger than or equal to the absolute deadline of the job generated by vertex  $u$ . In the real-time systems literature the first requirement is referred to as the *frame separation property* [10] and the second as the *localized Monotonic Absolute Deadlines property (l-MAD)* [3]. In this paper we assume either one of these two properties to hold.

**Task Sets and Schedulability Analysis.** A task set  $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$  consists of a collection of task graphs, the vertices of which can get triggered independently of each other. A triggering sequence for such a task set  $\mathcal{T}$  is legal if and only if for every task graph  $T_i$ , the subset of vertices of the sequence belonging to  $T_i$  constitute a legal triggering sequence for  $T_i$  (follows from the task independence assumptions in Section 2). In other words, a legal triggering sequence for  $\mathcal{T}$  is obtained by merging together (ordered by triggering times, with ties broken arbitrarily) legal triggering sequences of the constituting tasks.

The schedulability analysis of a task set  $\mathcal{T}$  is concerned with determining whether the jobs generated by all possible legal triggering sequences of  $\mathcal{T}$  can be scheduled such that their associated deadlines are met.

## 4 Algorithms for Approximate Schedulability Analysis

In this paper we will be only concerned with dynamic-priority schedulability analysis, and Theorem 1 in Section 2.1 applies to this class only. However, it may be noted that our general scheme for approximate schedulability analysis can be applied to static-priority schedulability analysis as well. We postpone the details of this to a full version of this paper, and here the term “schedulability analysis” always refers to dynamic-priority schedulability analysis only.

The schedulability analysis for the Recurring Real-Time task model presented in [2, 1] relies on the test given by Theorem 1. It checks the value of  $\sum_{T \in \mathcal{T}} T.dbf(t)$  for all values of  $t \leq t_{\max}$  and returns NOT SCHEDULABLE if for any value of  $t$  this sum exceeds  $t$ . It can be proved that  $t_{\max} = \frac{\sum_{T \in \mathcal{T}} 2E(T)}{1 - \sum_{T \in \mathcal{T}} \frac{E(T)}{P(T)}}$  which is therefore pseudo-polynomial in the size of the input specification of  $T$ . Here  $E(T)$  is the maximum cumulative execution requirement arising from a sequence of vertices on any path from the source to the sink vertex of the task graph of  $T$ .

For any task graph  $T$ , computing the value of  $T.dbf(t)$  for some values  $t \leq t_{\max}$  involves multiple traversals (loops) through the task graph. It was shown in [1] that if for a task graph  $T$ ,  $T.dbf(t)$  is known for all “small values” of  $t$  then it is possible to calculate from these, the value of  $T.dbf(t)$  for any  $t$ . “Small values” of  $t$  for a task graph  $T$  are those for which the sequence of vertices that contribute towards computing  $T.dbf(t)$  contain the source vertex at most once. The value of  $T.dbf(t)$  for larger values of  $t$  is made up of some multiple of  $E(T)$  plus  $T.dbf(t')$  where  $t'$  is “small” in the sense described above. It follows that  $T.dbf(t)$  for any  $t$  can be computed as follows (for a more detailed description, refer to [1])

$$T.dbf(t) = \max\{\lfloor t/P(T) \rfloor E(T) + T.dbf(t \bmod P(T)), (\lfloor t/P(T) \rfloor - 1)E(T) + T.dbf(P(T) + t \bmod P(T))\} \quad (2)$$

To compute  $T.dbf(t)$  for “small” values of  $t$ , [1] constructs a new task graph by taking two copies of the task graph of  $T$  and adding an edge from the sink vertex of the first graph to the source vertex of the second and finally replacing the source vertex of the first with a “dummy” vertex with execution requirement and deadline equal to zero. The intertriggering separations on all edges outgoing from this source vertex is also made equal to zero.  $T.dbf(t)$  for all values of  $t$  are then calculated by enumerating all possible paths in this new graph. In [6] it was shown that the problem of computing  $T.dbf(t)$  for a task  $T$  is NP-hard and an FPTAS for computing it was given. In the next section we describe this algorithm and show how it can be used for approximate schedulability analysis following the framework described in Section 2.2.

## 4.1 Approximating the Demand-Bound Function

The algorithm given in this section constitutes the *Building Block 1* shown in Figure 1. We first give an algorithm for computing the demand-bound function of a task graph for “small values” of  $t$ . Using this, we compute the demand-bound function for any value of  $t$  as described in Section 4.

Given a task graph  $T$ , let  $T'$  denote the graph formed by joining two copies of  $T$  by adding an edge from the sink vertex of the first graph to the source vertex of the second, and replacing the source vertex of the first copy by a “dummy” vertex as described above. If the frame separation property is followed then the newly added edge is labeled with an intertriggering separation of  $p = d(v_{sink})$ , and if the 1-MAD property is followed then it is labeled with  $p = \min\{0, d(v_{sink}) - d(v_{source})\}$ , where  $v_{source}$  and  $v_{sink}$  denotes the source and the sink vertices of  $T$ . Now we give a pseudo-polynomial time algorithm based on dynamic programming, for computing  $T'.dbf(t)$  for values of  $t$  that do not involve any looping through  $T'$ , i.e. we consider only “one-shot” executions of  $T'$ .

Let there be  $n$  vertices in  $T'$  denoted by  $v_1, \dots, v_n$ , and without any loss of generality we assume that there can be a directed edge from  $v_i$  to  $v_j$  only if  $i < j$ . Following our notation described in Section 3, associated with each vertex  $v_i$  is its execution requirement  $e(v_i)$  which here is assumed to be integral (a pseudo-polynomial algorithm is meaningful only under this assumption), and its deadline  $d(v_i)$ . Associated with each edge  $(v_i, v_j)$  is the minimum intertriggering separation  $p(v_i, v_j)$ .

Let  $t_{i,e}$  be the minimum time interval within which the task  $T'$  can have an execution requirement of exactly  $e$  time units due to some legal triggering sequence, considering only a subset of vertices from the set  $\{v_1, \dots, v_i\}$ , if all the triggered vertices are to meet their respective deadlines. Let  $t_{i,e}^i$  be the minimum time interval within which a sequence of vertices from the set  $\{v_1, \dots, v_i\}$ , and ending with the vertex  $v_i$ , can have an execution requirement of exactly  $e$  time units, if all the vertices have to meet their respective deadlines. Lastly, let  $E = \max_{i=1, \dots, n} e(v_i)$ . Clearly,  $nE$  is an upper bound on  $T'.dbf(t)$  for any  $t \geq 0$  for one-shot executions of  $T'$ . It can be trivially shown by induction that Algorithm 1 correctly computes  $T'.dbf(t)$ , and has a running time of  $O(n^3 E)$ .

Given this algorithm, any  $t \geq 0$ , and an  $0 < \varepsilon \leq 1$ , let  $T'_t$  be the subgraph of  $T'$  consisting only of those vertices  $v_i$  for which  $d(v_i) \leq t$ , and let  $E_t$  denote the maximum execution requirement of a vertex from among all vertices of  $T'_t$ . Now we scale all the execution requirements associated with the vertices of  $T'_t$  by  $K = \varepsilon E_t / n$  i.e.  $e'(v_i) = \lfloor e(v_i) / K \rfloor$  and run the algorithm with the new  $e'(v_i)$ s and the graph  $T'_t$ . Let  $V$  be the set of vertices (with the scaled execution requirements) that result in the computation of  $T'.dbf(t)$  in this algorithm. We claim that the summation of the orig-

---

## Algorithm 1 Computing $T'.dbf(t)$

---

**Input:** Task graph  $T'$ , and a real number  $t \geq 0$

```

for  $e \leftarrow 1$  to  $nE$  do
   $t_{1,e} \leftarrow \begin{cases} d(v_1) & \text{if } e(v_1) = e \\ \infty & \text{otherwise} \end{cases}$ 
   $t_{1,e}^1 \leftarrow t_{1,e}$ 
end for
for  $i \leftarrow 1$  to  $n - 1$  do
  for  $e \leftarrow 1$  to  $nE$  do
    Let there be directed edges from the vertices  $v_{i_1}, v_{i_2}, \dots, v_{i_k}$  to  $v_{i+1}$ 
     $t_{i+1,e}^{i+1} \leftarrow \begin{cases} \min\{t_{i_j,e-e(v_{i+1})}^j - d(v_{i_j}) + p(v_{i_j}, v_{i+1}) + \\ d(v_{i+1}) \mid j = 1, \dots, k\} & \text{if } e(v_{i+1}) < e, \\ d(v_{i+1}) & \text{if } e(v_{i+1}) = e, \text{ and } \infty \text{ otherwise} \end{cases}$ 
     $t_{i+1,e} \leftarrow \min\{t_{i,e}, t_{i+1,e}^{i+1}\}$ 
  end for
end for
 $T'.dbf(t) \leftarrow \max\{e \mid t_{n,e} \leq t\}$ 

```

---

inal (unscaled) execution requirements of these vertices is greater than or equal to  $(1 - \varepsilon)$  times the actual demand-bound function for the task graph for this value of  $t$ . Further, this algorithm now runs in time  $O(n^4/\varepsilon)$ , (with the scaled execution requirements), and hence is an FPTAS for computing  $T'.dbf(t)$ . We denote this approximate value of  $T'.dbf(t)$  computed by this algorithm by  $T'.dbf'(t)$ .

If a task graph  $T$  has  $O(n)$  vertices, then  $E(T)$  can be computed in  $O(n^2)$  time. Using the FPTAS given above,  $T.dbf(t)$  for any value of  $t$  can therefore be approximately computed using Equation 2 in  $O(n^4/\varepsilon)$  time. If we denote this approximate demand-bound function by  $T.dbf'(t)$  then for any  $t$ ,  $T.dbf(t) \geq T.dbf'(t) \geq (1 - \varepsilon)T.dbf(t)$ .

## 4.2 Checking the Sum of the Demand-Bound Functions

This section describes the *Building Block 2* in Figure 1 for the Recurring Real-Time task model. Recall from Section 4 that the exact algorithm for schedulability analysis for this model requires a pseudo-polynomial number of checks of the sum of the demand-bound functions. Following our framework for approximate schedulability analysis described in Section 2.2, we now show that combined with the approximate demand-bound function computed in the last section, a polynomial number of checks result in a bounded error. As in Section 2.2, we present an optimistic and a pessimistic algorithm, and in addition present a third algorithm where both SCHEDULABLE and NOT SCHEDULABLE answers can be wrong. However, we show that the worst case error incurred in either of these wrong decisions is less than the error incurred by the previous two algorithms.

The bounds on the error we obtain here for all the algorithms are tighter than the bounds derived in Section 2.2 for the abstract model. Given the FPTAS described in the last section for approximating the value of  $T.dbf(t)$ , it is possible to obtain the following bound on the approximate value of the demand-bound function:  $T.dbf'(t)$ :  $T.dbf'(t) + \varepsilon E_T \geq T.dbf(t)$ . Here  $E_T$  is the maximum execution requirement of any vertex in the task graph  $T$ .

---

**Algorithm 2** Optimistic Algorithm ( $t_{\max}, \delta$ )

---

**Input:**  $t_{\max}, \delta, m$  = number of task graphs in  $\mathcal{T}$ , for each task graph  $T$ ,  $E_T$  is the maximum execution requirement of any vertex in  $T$   
 $K \leftarrow \frac{\delta t_{\max}}{\text{poly}(m)}$   
 $error \leftarrow 0$   
 $decision \leftarrow$  SCHEDULABLE  
**for**  $t \leftarrow 1$  to  $\lfloor \frac{t_{\max}}{K} \rfloor + 1$  **do**  
  **if**  $\sum_{T \in \mathcal{T}} T.dbf'(tK) > tK$  **then**  
     $decision \leftarrow$  NOT SCHEDULABLE  
  **else**  
     $error\_in\_this\_interval \leftarrow$   
     $\max\{\min\{\frac{1}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(tK), \sum_{T \in \mathcal{T}} T.dbf'(tK) +$   
     $\varepsilon \sum_{T \in \mathcal{T}} E_T\} - (t-1)K, 0\}$   
     $error \leftarrow \max\{error, error\_in\_this\_interval\}$   
  **end if**  
**end for**  
 $return(decision, error)$

---

Further, since  $\frac{1}{1-\varepsilon}T.dbf'(t) \geq T.dbf(t)$ , we have

$$T.dbf(t) \leq \min\left\{\frac{1}{1-\varepsilon}T.dbf'(t), T.dbf'(t) + \varepsilon E_T\right\} \quad (3)$$

which therefore gives the better of the two bounds for any value of  $t$ .

For any task set  $\mathcal{T}$ , Algorithm 2 always returns the correct answer if  $\mathcal{T}$  is schedulable but might err if  $\mathcal{T}$  is not schedulable. Hence, whenever this algorithm returns NOT SCHEDULABLE, the decision is guaranteed to be correct. But SCHEDULABLE answers might be wrong. From Section 2.2, we obtain that the maximum possible error that can be incurred by the algorithm is equal to  $K + \frac{\varepsilon}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(\lfloor \frac{t_{\max}}{K} \rfloor + 1)K$ . However, using inequality (3) we can obtain a tighter bound on the error given by:  $K + \min\{\frac{\varepsilon}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(\lfloor \frac{t_{\max}}{K} \rfloor + 1)K, \varepsilon \sum_{T \in \mathcal{T}} E_T\}$ .

However, for any given problem instance, the maximum error that is incurred will possibly be lower than the theoretical worst case bound. Algorithm 2 computes this error for each problem instance. For this, consider any interval  $[iK, (i+1)K]$  such that  $\sum_{T \in \mathcal{T}} T.dbf'(t)$  is checked at  $t = iK$  and  $t = (i+1)K$ . If at  $t = (i+1)K$ , the computed upper bound on  $\sum_{T \in \mathcal{T}} T.dbf(t)$  (computed using inequality (3)) is less than or equal to  $iK$  then the error incurred in the interval  $(iK, (i+1)K]$  is equal to 0, since then for any  $t \in (iK, (i+1)K]$ ,  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is guaranteed to be less than or equal to  $t$ . Alternatively, if the computed upper bound on  $\sum_{T \in \mathcal{T}} T.dbf(t)$  is greater than  $iK$  then the maximum possible error in the interval  $[iK, (i+1)K]$  is equal to the difference between this bound and  $iK$ . Algorithm 2 computes this error at each interval and outputs the maximum among the computed error values.

Algorithm 3 gives the corresponding pessimistic algorithm in which SCHEDULABLE answers are guaranteed to be correct and NOT SCHEDULABLE answers might be wrong, where the maximum error is the same as in the case of Algorithm 2. Finally, Algorithm 4 uses an upper bound on the value of  $T.dbf(t)$  as given by inequal-

---

**Algorithm 3** Pessimistic Algorithm ( $t_{\max}, \delta$ )

---

**Input:**  $t_{\max}, \delta, m$  = number of task graphs in  $\mathcal{T}$ , for each task graph  $T$ ,  $E_T$  is the maximum execution requirement of any vertex in  $T$  and  $d_T$  is the minimum deadline associated with any vertex in  $T$   
 $K \leftarrow \frac{\delta t_{\max}}{\text{poly}(m)}$   
 $decision \leftarrow$  SCHEDULABLE  
**for**  $t \leftarrow 1$  to  $\lfloor \frac{t_{\max}}{K} \rfloor + 1$  **do**  
  **if**  $\min\{\frac{1}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(d_T + tK), \sum_{T \in \mathcal{T}} T.dbf'(d_T + tK) +$   
   $\varepsilon \sum_{T \in \mathcal{T}} E_T\} > d_T + (t-1)K$  **then**  
     $decision \leftarrow$  NOT SCHEDULABLE  
  **end if**  
**end for**  
 $return(decision)$

---

---

**Algorithm 4** Algorithm with double sided error

---

**Input:**  $t_{\max}, \delta, m$  = number of task graphs in  $\mathcal{T}$ , for each task graph  $T$ ,  $E_T$  is the maximum execution requirement of any vertex in  $T$   
 $K \leftarrow \frac{\delta t_{\max}}{\text{poly}(m)}$   
 $decision \leftarrow$  SCHEDULABLE  
**for**  $t \leftarrow 1$  to  $\lfloor \frac{t_{\max}}{K} \rfloor + 1$  **do**  
  **if**  $\min\{\frac{1}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(tK), \sum_{T \in \mathcal{T}} T.dbf'(tK) +$   
   $\varepsilon \sum_{T \in \mathcal{T}} E_T\} > tK$  **then**  $decision \leftarrow$  NOT SCHEDULABLE  
  **end if**  
**end for**  
 $return(decision)$

---

ity (3), along with the optimistic version of the checking procedure given by our *Building Block 2* in Figure 1. This algorithm incurs a double sided error—SCHEDULABLE answers can be wrong, but the maximum error in this case is bounded by  $K$ . NOT SCHEDULABLE answers can be wrong too, but the error in this case is bounded by  $\min\{\frac{\varepsilon}{1-\varepsilon} \sum_{T \in \mathcal{T}} T.dbf'(\lfloor \frac{t_{\max}}{K} \rfloor + 1)K, \varepsilon \sum_{T \in \mathcal{T}} E_T\}$ . Hence, the maximum error in either case is smaller than the maximum error incurred by the optimistic and the pessimistic algorithms.

**Running Times.** It can be shown that computing all possible values of  $T.dbf'(t)$  for “small values” of  $t$  and for all task graphs  $T \in \mathcal{T}$ , takes  $O(n^5 m/\varepsilon)$  time, where each task graph in  $\mathcal{T}$  contains  $O(n)$  vertices and  $\mathcal{T}$  contains  $m$  task graphs [6]. All of these values are first computed and stored in a table. During the second phase of the algorithm (when the sum of the demand-bound functions are checked to determine if they exceed  $t$ ), for any  $t$ , computing the value of  $\sum_{T \in \mathcal{T}} T.dbf'(t)$  needs table lookup which takes  $O(n^2 m \varepsilon^{-1} \log n)$  time. Since the sum of the demand bound functions is checked for  $O(\frac{\text{poly}(m)}{\delta})$  values of  $t$ , the running time of any of the three algorithms described above is  $O(n^5 m \varepsilon^{-1} + n^2 m \varepsilon^{-1} \delta^{-1} \text{poly}(m) \log n)$  time. Hence, our approximate schedulability analysis for the Recurring Real-Time task model runs in polynomial time.

## 5 Experimental Results

This section reports some experimental results obtained by applying Algorithm 2 on a set of randomly generated task graphs. We have implemented an exact schedulability analysis algorithm, based on the dynamic programming algorithm (Algorithm 1) for computing the demand-bound

		$\delta \rightarrow$				
		exact	0.2	0.4	0.6	0.8
$\varepsilon$ ↓	exact	775	36	18	11	9
	0.2	781	35	17	11	9
	0.4	759	34	17	11	8
	0.6	749	34	16	11	8
	0.8	728	33	16	11	8

**Table 1.** Running times (in ms) of *Building Block 2* in

Figure 1 for  $K = \frac{\delta t_{\max}}{m^6}$

function and testing if the sum of the demand-bound functions for all the task graphs exceed  $t$  for all  $t \leq t_{\max}$ . This algorithm runs in pseudo-polynomial time. We compare the running time of this algorithm against the algorithm for approximate schedulability analysis.

For our experiments we have randomly generated synthetic task graphs, using two parameters. The first is the maximum execution requirement,  $E$ , associated with any vertex of the graph. The running time of the pseudo-polynomial exact algorithm and the quality of the results obtained by the approximate schedulability analysis, both, depend on this parameter. We call the second parameter the *connectivity factor*. If  $v_1, \dots, v_n$  are the vertices of a task graph such that there is an edge from  $v_i$  to  $v_j$  only if  $j > i$ , then while generating the graph for each vertex  $v_j$  we construct an edge from  $v_i$  to  $v_j$  with a probability equal to the connectivity factor of the graph, for all  $i = 1, \dots, j - 1$ .

Our randomly generated task graphs represent a network packet processing case study. The parameters used to generate these graphs (i.e.  $E$  and the connectivity factor) reflect this application domain for our algorithms. Details of these application may be found in [5]. For example, a connectivity factor equal to 0.4 is used in all the tests since this results in graphs which are similar to those arising in practice, and  $E$  is either equal to 200 or 600 representing two possible cases in the above mentioned application.

Figures 2(a) and 2(b) show the running times involved in computing the demand bound function for a single task graph using the exact pseudo-polynomial algorithm (i.e. Algorithm 1) and the FPTAS for four different values of  $\varepsilon$ , when the number of vertices in the graphs is gradually increased. In the first figure, the maximum execution requirement of a vertex is set to 200, while in the second it is equal to 600. Note that in the former case, the exact algorithm is better than the FPTAS for  $\varepsilon = 0.2$  when the number of vertices in the graph increase beyond 40. Therefore, the optimal choice of  $\varepsilon$  depends on  $E$  and the number of vertices in the task graph.

In Table 1 we show the running times involved in performing the test to determine if the sum of the demand-bound functions for any  $t$  exceeds  $t$ , for exact case where  $t_{\max}$  tests are performed and for different values of  $\delta$  in the approximate analysis. Here the task set consists of three task graphs with 30 vertices per graph and the maximum execution requirement associate with any vertex is equal to 200, and  $K = \frac{\delta t_{\max}}{m^6}$  (where  $m = 3$  is the number of tasks

in the task set). Note that the times reported here do not involve the time required to compute the demand-bound functions. These are precomputed and stored in a table.

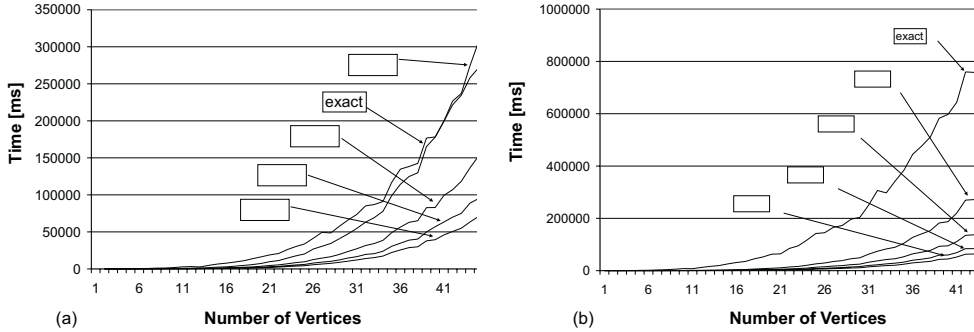
Lastly, in Figure 3(a) we show the percentage of wrong answers returned by the algorithm for different values of  $\varepsilon$  and  $\delta$ . For this we have averaged the results obtained from a set of 600 task sets, each, as before, consisting of three task graphs with 30 vertices in each graph and maximum execution requirement of any vertex equal to 200. Here, again  $K = \frac{\delta t_{\max}}{m^6}$  and  $m = 3$ . Figure 3(b) shows the corresponding maximum *error* values obtained, averaged over the 600 task sets, for different values of  $\varepsilon$  and  $\delta$ .

Instead of  $m^6$ , if a higher degree polynomial is used, then the maximum error incurred and the percentage of wrong results, both decrease because more values of  $t$  are then tested. All the task sets considered in the experiment were generated such that they either fully load the processor, or when not schedulable, the vertices miss their deadlines only by small lengths of time. Therefore, these represent the most difficult cases for the approximate schedulability analysis algorithm. If the task sets are either “comfortably” schedulable, leaving the processor idle over long intervals of time, or if they overload the processor in the sense that jobs miss their deadlines by large time lengths, then the percentage of wrong results returned by the approximate schedulability analysis algorithm decrease. If the *load-factor* of a task set  $\mathcal{T}$  is defined as  $\max\{t = 1, \dots, t_{\max} \mid \frac{\sum_{T \in \mathcal{T}} T.dbf(t)}{t}\}$ , then the load-factor of all the task sets considered in our experiments were between 0.93 and 1.11. Lastly, among the 600 task sets considered, around 50% were schedulable and the rest not.

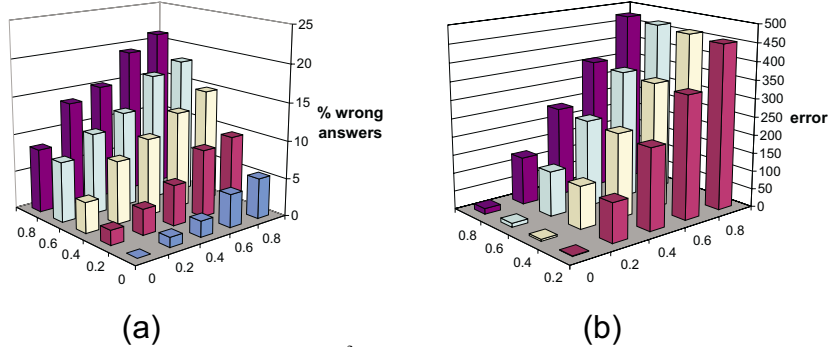
All the CPU times reported here were measured on a moderately loaded Sunblade 1000 running SunOS 5.8 with 750 MHz CPU and 2 GB RAM. All the algorithms were implemented in Java.

## 6 Other Task Models

Our results in Section 2.2 also imply polynomial time algorithms for approximate schedulability analysis for models such as the Sporadic, Multiframe and the Generalized Multiframe. All of these models can be considered to be special cases of the Recurring Real-Time task model. Nevertheless, we especially point them out here since the resulting algorithms for approximate schedulability analysis are considerably simpler compared to those for the Recurring Real-Time task model. The reason for this is that, in all of these models the demand-bound function for any  $t$  can be computed (exactly) in polynomial time. Therefore, the pseudo-polynomial running times of the known algorithms for schedulability analysis for all of these models are attributed to the pseudo-polynomial number of tests to verify if for any  $t$  the sum of the demand-bound functions exceed  $t$ . If  $t_{\max}$  is the number of tests to be performed, then as shown in Section 2.2, an algorithm for approximate schedulabil-



**Figure 2.** Running times for computing the demand-bound function for a single task graph with (a)  $E = 200$  and (b)  $E = 600$ .



**Figure 3.** (a) Percentage of wrong answers with  $K = \frac{\delta t_{\max}}{m\theta}$  for different values of  $\epsilon$  and  $\delta$ . (b) Maximum error incurred by the decisions returned by the schedulability analysis algorithm, averaged over 600 randomly generated task sets.

ity analysis for any of these models performs only  $O(\frac{t_{\max}}{K})$  tests where  $K$ , as before, is equal to  $\frac{\delta t_{\max}}{\text{poly}(|T|)}$  ( $\delta$  is the input error parameter to the algorithm). The error, in the case of both optimistic and pessimistic algorithms is bounded by  $K = \frac{\delta t_{\max}}{\text{poly}(|T|)}$ . Since the demand-bound function for any  $t$  in these models can be computed in polynomial time, and the number of tests as mentioned of  $O(\frac{\text{poly}(|T|)}{\delta})$ , it implies polynomial time algorithms for approximate schedulability analysis. Due to space constraints here we skip the details of the algorithms and the exact value of  $K$  for the different models, and postpone them to a full version of this paper.

## 7 Concluding Remarks

In this paper we introduced a notion of approximate schedulability analysis and showed that for number of well known task models the schedulability analysis problem can be solved in polynomial time, provided some “error” in the decisions made by the algorithm is acceptable. We further showed that this error can be bounded, and also that the amount of error incurred can be controlled by an input error parameter  $(\epsilon, \delta)$ . Here  $(\epsilon, \delta)$  represented a tradeoff between the running time of the algorithm and the error incurred.

The problem of designing efficient *false-negative* schedulability tests for the Generalized Multiframe task model was stated as a possible future research direction in [3]. Our work addresses a much more generalized version of this problem. Apart from being of theoretical interest, the experimental results suggest that the algorithms are implementable and lead to clear improvements in terms of running time.

## References

- [1] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. To appear in *Real-Time Systems*.
- [2] S. Baruah. A general model for recurring real-time tasks. In *Proc. 19th IEEE Real-Time Systems Symposium*, pages 114–122. IEEE Computer Society Press, 1998.
- [3] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.
- [4] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.
- [5] S. Chakraborty, T. Erlebach, S. Künzli, and L. Thiele. Schedulability of event-driven code blocks in real-time embedded systems. In *Proc. 39th Design Automation Conference (DAC)*. ACM Press, 2002.
- [6] S. Chakraborty, T. Erlebach, and L. Thiele. On the complexity of scheduling conditional real-time code. In *Proc. 7th International Workshop on Algorithms and Data Structures (WADS)*, Lecture Notes in Computer Science 2125, pages 38–49, 2001.
- [7] D. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.
- [8] A. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, MIT, 1983. Available as Technical Report No. MIT/LCS/TR-297.
- [9] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.
- [10] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *Proc. 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, pages 80–86, 1997.