Utilization Bound Revisited

Deji Chen, Aloysius K. Mok, and
Tei-Wei Kuo, *Member*, *IEEE Computer Society*

Abstract—Utilization bound is a well-known concept introduced in the seminal paper of Liu and Layland, which provides a simple and practical way to test the schedulability of a real-time task set. The original utilization bound for the fixed-priority scheduler was given as a function of the number of tasks in the periodic task set. In this paper, we define the utilization bound as a function of the information about the task set. By making use of more than just the number of tasks, better utilization bound over the Liu and Layland bound can be achieved. We investigate in particular the bound given a set of periods for which it is still unknown if there is a polynomial algorithm for the exact bound. By investigating the relationships among the periods, we derive algorithms that yield better bounds than the Liu and Layland bound and the harmonic chain bound. Randomly generated task sets are tested against different bound algorithms. We also give a more intuitive proof of the harmonic chain bound and derive a computationally simpler algorithm.

Index Terms—Preemptive fixed-priority scheduling, rate-monotonic priority assignment, utilization bound.

1 Introduction

Despite the tremendous amount of work and advances in real-time scheduling theory, the best results for schedulability testing of periodic task sets under the preemptive fixed priority scheduling policy provide only pseudopolynomial time algorithms. For periodic tasks whose deadlines are before their periods, the feasibility problem is co-NP-hard [15] if the initial requests can arrive at different points in time. In practice, pseudopolynomial-time algorithms are often acceptable as compile-time (offline) schedulability tests, as has been popularized in RMA (Rate Monotonic Analysis) [11]. As real-time applications grow in complexity, task sets become more dynamic in that their parameters may change at runtime. For these applications, the pseudopolynomial-time schedulability tests may no longer be fast enough for use as admission tests in real time. One approach is deriving sufficient but not necessary polynomial testing algorithms [9]. For a task set, if the test succeeds, then the task set is schedulable; if the test fails, then the task set may or may not be schedulable. In contrast, the concept of utilization bound provides a fast sufficiency test for schedulability. The classic Liu and Layland paper [16] proves that, for a set of n periodic tasks whose deadlines are the same as their periods, the utilization bound under the preemptive fixed-priority scheduling policy and the rate monotonic priority assignment is $n(2^{\frac{1}{n}}-1)$. Several other improvements and extensions on the utilization bound have been reported since then [3], [12], [14], [17], [20].

We shall show that the utilization bound of a task set can be more precisely determined if more information about the task set is exploited. The Liu and Layland bound $n(2^{\frac{1}{n}}-1)$ depends only on the number of tasks n in the task set [16]. It can be improved to $K(2^{\frac{1}{K}}-1)$, where K, $K \leq n$, is the number of harmonic chains in the task set. A harmonic chain is a list of numbers in which every number divides every number after it. We shall give a more intuitive derivation of this bound, which allows us to derive a simpler algorithm than determining the number of harmonic chains for a given task set. If we make use of the period parameters in the task set, even better bounds can be derived. For example, [3] proved that a better bound can be achieved if all periods in a task set have values that are close to each other. Of course, if we make use of the execution times of the tasks as well, we can tell exactly if the task set is schedulable or not by performing a pseudopolynomial-time test, e.g., the critical-instant test as formulated in [2], [4], [10]. As we have mentioned, this may not be practical as an online admission test for dynamic real-time systems. There are different necessary and sufficient utilization bounds (exact bounds) for different classes of task sets. The more information of the task sets is provided, the better the utilization bound of the task sets. In this paper, we look at the task model where relative deadline equals the period and investigate the bound given the array of task periods. Even though we consider only periodic tasks which start requesting at time 0, the results in this paper can be easily extended to sporadic tasks which have arbitrary request times and minimum separation times. In addition, we shall assume rate/deadline-monotonic assignment of priority and the use of the critical-instant test as a feasibility test throughout this paper.

The paper is organized as follows: Section 2 introduces the concepts of *extremely* and *critically* utilizing task set and describes our approach to find the utilization bound.

- *D. Chen is with the Austin Technology Center, Schlumberger, 8311 North FM 620 Rd., Austin, TX 78726. E-mail: cdj@cs.utexas.edu.*
- *A.K. Mok is with the Department of Computer Sciences, College of Natural Sciences, University of Texas at Austin, Austin, TX 78712. E-mail: mok@cs.utexas.edu.*
- *T.-W. Kuo is with the Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Sec 4 Rossevelt Rd., Taipei, Taiwan 106, ROC. E-mail: ktw@csie.ntu.edu.tw.*

Manuscript received 24 Jan. 2001; revised 12 July 2001; accepted 17 Sept. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 113529.

Section 3 looks for the exact utilization bound given task periods. We believe, in general, the exact bound is difficult to get and instead try to derive some bound algorithms in Section 4. Park et al. [18], [19] derived this exact bound by linear programming, which is a novel approach and is not polynomial. Section 5 tests these algorithms with randomly generated task sets. We shall see how close they are to the exact bound. Finally, Section 6 is the conclusion. We end this section with above-mentioned two bounds:

Theorem 1 (Theorem 5 in [16]). *For a set of n tasks with fixed priority order, the least upper bound to processor utilization is* $U = n(2^{\frac{1}{n}} - 1)$.

Theorem 2 (Theorem 4 in [12]). *The least upper bound of the utilization factor for any task set in which the periods are selected from K harmonic chains is at least* $K(2^{\frac{1}{K}} - 1)$.

2 The Exact Bound for (E, P)-Type Task Set

An (E, P)-type task set is a set of periodic tasks $S = \{T_1, T_2, \ldots, T_n\}$, where $T_i = (E_i, P_i)$, $1 \leq i \leq n, E_i \leq P_i$, and E_i, P_i denote, respectively, the execution time and period of the task T_i. The relative deadline equals the period. The utilization factor U of a set S of n tasks is $\Sigma_{i=1}^{n} \frac{E_i}{P_i}$. Without loss of generality, we shall assume $P_1 < P_2 < \cdots < P_n$. By the rate monotonic assignment, T_1 has the highest priority and T_n has the lowest priority. The task periods will be represented by the vector $\vec{P} = [P_1, \ldots, P_n]$.

Definition 1. *A utilization bound* $\overline{U}$ *of real-time task sets is the value such that:*

- *Any task set whose utilization factor is no larger than* $\overline{U}$ *is schedulable by RMA assignment of priorities.*

Consider the class M of all (E, P)-type task sets that cannot be scheduled by the preemptive fixed-priority scheduler. Let $\overline{U}_{\mathrm{EP}}$ be the "greatest lower bound" of the utilization factors of all such task sets in M. Any task set that has a utilization factor smaller than $\overline{U}_{\mathrm{EP}}$ is by definition schedulable by the RMA assignment of task priority. This is the basis of the utilization-bound based schedulability test. We call $\overline{U}_{\mathrm{EP}}$ the exact utilization bound and state its property formally below:

Definition 2. *The exact utilization bound* $\overline{U}_{\mathrm{EP}}$ *of real-time task sets is the value such that:*

- Sufficient Condition: *Any task set whose utilization factor is no larger than* $\overline{U}_{\mathrm{EP}}$ *is schedulable by RMA assignment of priorities.*
- Necessary Condition: *For any value U, if* $U > \overline{U}_{\mathrm{EP}}$, *then there exists a task set whose utilization factor equals U and which is not schedulable by the RMA assignment of priority.*

Note that any value less than $\overline{U}_{\mathrm{EP}}$ can be a utilization bound; any utilization bound cannot be larger than $\overline{U}_{\mathrm{EP}}$. Note also that $\overline{U}_{\mathrm{EP}}$ is, in essence, the same as the "least upper bound of the utilization factor" referred to in [16].

At this point, we should emphasize that the definition of the exact utilization bound depends on the class of task sets under discussion. In the derivation of the Liu and Layland bound, the authors considered all task sets of size n and derive a function of n which is the exact utilization bound. To make this explicit, we shall make the exact utilization bound as a function of M—the class of task sets of interest —and we shall write the utilization bound $\overline{U}_{\mathrm{EP}}(M)$ as a function of M. For the Liu and Layland bound, we define $M_1 = \{S| \ \|S\| = n\}$, i.e., M_1 refers to all size-n task sets. For any $U > n(2^{\frac{1}{n}} - 1)$, we can form an unschedulable n-task set whose utilization is U. We have the following corollary [16]:

Corollary 1. $\overline{U}_{\mathrm{EP}}(M_1) = n(2^{\frac{1}{n}} - 1)$.

In the following, we shall consider $\overline{U}_{\mathrm{EP}}(M_2)$, where M_2 is defined in the following:

Definition 3. *For an array* $\vec{P}$ *of size n, define*

$$M_2 = \{S = \{T_1 = (E_1, P_1), T_2 = (E_2, P_2), \ldots, T_n = (E_n, P_n)\} | [P_1, P_2, \ldots, P_n] = \vec{P}\}.$$

Note we could define different class types of task sets. For example, if we define M' to be the class of size-n task sets whose ratio of the largest period to the smallest period r is less than 2, [13] proves that $\overline{U}_{\mathrm{EP}}(M') = (n-1)(r^{\frac{1}{n-1}} - 1) + \frac{2}{r} - 1$. We only concentrate on $\overline{U}_{\mathrm{EP}}(M_2)$ in this paper. Let us first take a look at an example.

Example 1. Consider a set of six tasks with periods forming a vector $\vec{P} = [2, 3, 5, 6, 7, 35]$. $\overline{U}_{\mathrm{EP}}(M_1) = 0.7348$ according to Corollary 1. $\vec{P}$ can be formed from four harmonic chains $(2, 6)$, (3), $(5, 35)$, and (7). The harmonic chain bound according to Theorem 2 is 0.7568. Assuming that the execution time can only be integers, we find that $\overline{U}_{\mathrm{EP}}(M_2) = 0.7952$ by checking all possible execution assignments.

Note that $\overline{U}_{\mathrm{EP}}(M_2)$ is larger than both bounds calculated in Example 1. The question is whether we can find an algorithm to calculate a better utilization bound or, even better, the exact bound. We do not at present have a closed-form expression for $\overline{U}_{\mathrm{EP}}(M_2)$. Instead, by investigating the properties of $\overline{U}_{\mathrm{EP}}(M_2)$, we shall find safe utilization bounds that lie between $\overline{U}_{\mathrm{EP}}(M_2)$ and $\overline{U}_{\mathrm{EP}}(M_1)$ or $K(2^{\frac{1}{K}} - 1)$. We start by defining two important concepts in Section 2.1 which are crucial for establishing our results.

2.1 Extremely and Critically Utilizing Task Sets

Definition 4. *A task set extremely utilizes a processor if:*

- *It is schedulable by RMA assignment of priorities.*
- *The execution time of the longest-period task is nonzero.*
- *Increasing the execution time of the longest-period task in the set will render the resulting task set unschedulable by RMA.*

We shall call a task set that extremely utilizes a processor an extreme task set.

Definition 5. *A task set critically utilizes a processor if:*

- *It is schedulable by the RMA assignment of priorities.*
- *Increasing the execution time of some task in the set will render the resulting task set unschedulable by RMA.*

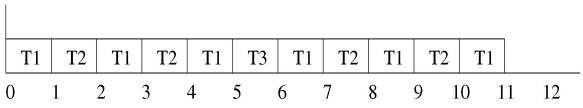


Fig. 1. The schedule of task set S.

We shall call a task set that critically utilizes a processor a critical task set.

Example 2. The task set $S = \{T_1, T_2, T_3\}$ with $T_1 = (1, 2), T_2 = (1, 3), T_3 = (1, 12)$ critically but not extremely utilizes the processor.

First, the task set S in Example 2 is schedulable as we can easily check its critical-instant. If we increase the execution time of either T_1 or T_2, S will be unschedulable; but we can increase the execution time of T_3 from 1 to 2, still resulting in a schedulable task set. The subset $\{T_1, T_2\}$ of S extremely utilizes the processor. Fig. 1 is the schedule of S in the first 12 time units.

Following the definition, we have:

Corollary 2. *An extreme task set is a critical task set.*

The critical instant of a task is when it requests simultaneously with all higher priority tasks.

Corollary 3. *If task set S extremely utilizes the process, the processor will not idle in the critical-instant test. If the task set is schedulable and the processor idles during the critical-instant test of the lowest-priority task (T_n), then the task does not extremely utilize the processor.*

Corollary 4. *Suppose task set S having utilization factor U critically but not extremely utilizes the processor. We can form a new task set S' with smallest possible j tasks where $j < n$ and $P'_i = P_i, E'_i = E_i$ for $1 \le i \le j$. S' extremely utilizes the processor and its utilization factor $U' \le U$.*

In Corollary 4, j is the smallest so that the increase of T_j's execution will render T_j unschedulable. We shall refer to S' in Corollary 4 as the derived extreme task set of S and denote it by S_{derived}.

Lemma 1. $\overline{U}_{\text{EP}} =$ *the minimum of the utilization factors of all critical task sets.*

Proof. Let $U_{\min} =$ the minimum of the utilization factors of all critical task sets and let $S_{\min}$ be a critical task set with utilization equal to $U_{\min}$ such that increasing E_i of the ith task $T_i = (E_i, P_i)$ in $S_{\min}$ will render $S_{\min}$ unschedulable.

1. For any U, if $U > U_{\min}$, we add $(U - U_{\min}) \times P_i$ to E_i. By construction, we get a new unschedulable task set with utilization factor U. By Definition 2, $U > \overline{U}_{\text{EP}}$. So, $U_{\min} \ge \overline{U}_{\text{EP}}$.
2. For any U, if $U > \overline{U}_{\text{EP}}$, since $\overline{U}_{\text{EP}}$ is an exact bount, we can find a task set S whose utilization factor U_S is greater than $\overline{U}_{\text{EP}}$ but no greater than U and it is also unschedulable. We can select a task from S which is unschedulable and reduce its execution time until it is just schedulable. By repeating this step until no task is unschedulable, we get a new task set whose utilization factor U'_S is less than U_S and whose latest selected task cannot increase its execution time without causing unschedulability. By definition, the derived task set critically utilizes processor. Thus, we have $U'_S \ge U_{\min}$. So, $U \ge U_S > U'_S \ge U_{\min}$. So, $\overline{U}_{\text{EP}} \ge U_{\min}$.

From the above 1 and 2, we get $\overline{U}_{\text{EP}} = U_{\min}$. □

Lemma 1 applies to all task set classes.

According to Lemma 1, the bound is the minimum of the utilization of all critical task sets. Suppose the minimum occurs with task set

$$S = \{T_1 = (E_1, P_1), T_2 = (E_2, P_2), \ldots, T_n = (E_n, P_n)\}$$

and the derived extreme task set is $S_{\text{derived}} = \{T_1, T_2, \ldots, T_m\}, m \le n$ (let $S_{\text{derived}} = S$ if S extremely utilizes processor). S_{derived} exists according to Corollary 4. We claim that:

- $E_j = 0$ if $j > m$.

Proof. If, for some $j > m$, we have $E_j > 0$, we can reduce E_j to 0 to get a new task set S', S' is still critical as increasing execution time of E_m will cause T_m unschedulable. Since S' has $\frac{E_j}{P_j}$ less utilization factor than that of S, it contradicts that S has the minimum utilization factor. So, $E_j = 0$. □

- The utilization factor of S_{derived} is the minimum of those of all extreme task sets with the same task number and periods as S_{derived}.

Proof. If another extreme task set

$$S'_{\text{derived}} = \{(E'_1, P_1), (E'_2, P_2), \ldots, (E'_m, P_m)\}$$

has utilization factor less than that of S_{derived}, we have a critical task set

$$S'_{\text{derived}} \bigcup \{(0, P_{m+1}), \ldots, (0, P_n)\}$$

that has smaller utilization factor than that of S. This is a contradiction. □

Theorem 3. *For $\vec{P} = [P_1, P_2, \ldots, P_n]$, $\overline{U}_{\text{EP}}(M_2) = \min_{i=1}^{n} \overline{U}_i$, where $\overline{U}_i$ is the minimum utilization factor of all extreme task sets of $\vec{P}^i = [P_1, P_2, \ldots, P_i]$.*

Proof. This follows directly from the above claims. □

With Theorem 3 in mind, we shall concentrate our attention on the minimum of all extreme task sets satisfying a given task period array $\vec{P}$.

3 Calculate the Exact Utilization Bound

In general, we could not find a polynomial algorithm for the exact bound of any $\vec{P}$. In this section, we give results for some special cases. Section 3.1 gives the bound when $P_n < 2P_1$, Section 3.2 finds the bound when $n = 2$,

Section 3.3 finds the bound when $n = 3$, and Section 3.4 looks at the general case.

3.1 $\overline{U}_{\mathrm{EP}}(M_2)$ When $P_n < 2P_1$

Lemma 2. *For $\vec{P}$, if $P_1 < P_2 < \ldots < P_n < 2P_1$, $\overline{U}_n$ occurs when:*

$$E_i = P_{i+1} - P_i,\ 1 \le i < n$$
$$E_n = 2P_1 - P_n$$
$$\text{and } \overline{U}_n = \Sigma_{i=1}^{n-1} \frac{P_{i+1} - P_i}{P_i} + \frac{2P_1 - P_n}{P_n}.$$

Proof. Let $\overline{U}_n$ happen with task set S, $S = \{T_1, T_2, \ldots, T_n\}$.

First, we can prove that, starting from time 0, the second request of any T_i with $1 \le i < n$ should be completed before P_n. Otherwise, we can assign part of E_i to E_n and derive an extreme task set with less utilization.

Second, we can prove that E_n should be completed exactly at P_1. Otherwise, we can assign part of E_n to another task and derive an extreme task set with less utilization. This implies that the first requests of all tasks finish before P_1.

Third, we can prove that the second request of $T_i, i < n$ should be completed exactly at time P_{i+1}. Otherwise, we can assign part of E_i to E_{i+1} and derive an extreme task set with less utilization.

Thus, we have:

$$\begin{aligned} E_i &= P_{i+1} - P_i,\ 1 \le i < n \\ E_n &= P_1 - (E_1 + E_2 + \ldots + E_{n-1}) \\ &= P_n - 2(E_1 + E_2 + \ldots + E_{n-1}) \\ &= 2P_1 - P_n. \end{aligned}$$

We can check that this assignment of execution time extremely utilizes the processor.

$$\overline{U}_n = \Sigma_{i=1}^{n-1} \frac{P_{i+1} - P_i}{P_i} + \frac{2P_1 - P_n}{P_n}.$$

□

Theorem 4. *For $\vec{P}$, if $P_1 < P_2 < \ldots < P_n < 2P_1$, $\overline{U}_{\mathrm{EP}}(M_2)$ is attained when:*

$$E_i = P_{i+1} - P_i,\ 1 \le i < n$$
$$E_n = 2P_1 - P_n$$
$$\text{and } \overline{U}_{\mathrm{EP}}(M_2) = \Sigma_{i=1}^{n-1} \frac{P_{i+1} - P_i}{P_i} + \frac{2P_1 - P_n}{P_n}.$$

Proof. According to Theorem 3 and Lemma 2, we only need to prove that $\overline{U}_{\mathrm{EP}}$ is attained at $\overline{U}_n$. We prove this by contradiction. Suppose $\overline{U}_{\mathrm{EP}} = \overline{U}_i$, where $i < n$. According to Lemma 2, we have the task set S that has the bound as its utilization factor:

$$E_j = P_{j+1} - P_j,\ 1 \le j < i,$$
$$E_i = 2P_1 - P_i,$$

and $E_{i+1} = E_{i+2} = \cdots = E_n = 0$.

Since $E_i = 2P_1 - P_i > P_n - P_i$, we know part of the second request of T_2 finishes after P_n. Let us say this part is Δ; we can derive a new task set S' by decreasing E_i with Δ and increasing E_n with Δ.

S' critically utilizes the processor since we cannot increase E'_n without rendering T'_n unschedulable. $U' = \overline{U}_i - \frac{\Delta}{P_i} + \frac{\Delta}{P_n} = \overline{U}_i + \Delta \times \frac{P_i - P_n}{P_i P_n} < \overline{U}_i$.

This contradicts the assumption that S has the smallest utilization factor. Hence, $\overline{U}_{\mathrm{EP}} = \overline{U}_n$ and the theorem is proven. □

3.2 $\overline{U}_{\mathrm{EP}}(M_2)$ When $n = 2$

Theorem 5. *For a two-task set of $\vec{P} = [P_1, P_2], P_1 < P_2$, let $P_2 = p_1 * P_1 + r_1, 0 \le r_1 < P_1$. $\overline{U}_{\mathrm{EP}}(M_2)$ is obtained when:*

$$E_1 = r_1$$
$$E_2 = P_2 - (p_1 + 1)r_1$$
$$\text{and } \overline{U}_{\mathrm{EP}}(M_2) = \frac{E_1}{P_1} + \frac{E_2}{P_2}.$$

Proof. $\overline{U}_{\mathrm{EP}}(M_2) = \min(\overline{U}_1, \overline{U}_2)$. It is obvious that $\overline{U}_1 = 1$. The correctness for $\overline{U}_2$ can be proven the same as the proof of Theorem 3 in [16]. It is obvious that $\overline{U}_2 \le 1 = \overline{U}_1$, the execution assignment in the theorem also achieves $\overline{U}_{\mathrm{EP}}(M_2)$. □

3.3 $\overline{U}_{\mathrm{EP}}(M_2)$ When $n = 3$

It turns out the exact bound for a 3-task set is much more complicated than a 2-task set.

Look at a 3-task set of $\vec{P} = [P_1, P_2, P_3], P_1 < P_2 < P_3$. If P_1 divides P_2 or P_3, or P_2 divides P_3, $\overline{U}_{\mathrm{EP}}(M_2)$ can be reduced to that of a 2-task set. So, we shall now consider the case where all periods are prime to each other.

Let $P_3 = p_1 * P_1 + r_1 = P_{1m} + r_1, 0 < r_1 < P_1$; let

$$P_3 = p_2 * P_2 + r_2 = P_{2m} + r_2, 0 < r_2 < P_2.$$

According to Theorem 3, $\overline{U}_{\mathrm{EP}}(M_2) = \min(\overline{U}_1, \overline{U}_2, \overline{U}_3)$. Since $\overline{U}_1 = 1$, $\overline{U}_{\mathrm{EP}}(M_2) = \min(\overline{U}_2, \overline{U}_3)$. $\overline{U}_2$ can be calculated with Theorem 5. We shall now consider $\overline{U}_3$.

Assume $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3$ and look at a task set $S = \{T_1, T_2, T_3\} = \{(E_1, P_1), (E_2, P_2), (E_3, P_3)\}$ with utilization $\overline{U}_3$. We claim that, in its critical-instant:

- Both requests of T_1 at P_{1m} and T_2 at P_{2m} finish before P_3.

Proof. Suppose δ of T_1 finishes after P_3, we could reduce E_1 by δ and increase E_3 by $p_1\delta$. We still have an extreme task set but utilization factor less than $\overline{U}_{\mathrm{EP}}(M_2)$. This is not possible. So, T_1's request at P_{1m} finishes before P_3. So does T_2's request at P_{2m}. □

- T_3 finishes before P_{1m} and P_{2m}.

Proof. Suppose δ of T_3 finishes after P_{1m}, we could reduce E_3 by δ and increase E_1 by $\frac{\delta}{p_1+1}$. We either have an unschedulable or extreme task set but utilization factor less than $\overline{U}_{\mathrm{EP}}(M_2)$. This is not possible. So, T_3 finishes before P_{1m}. For the same reason, T_3 finishes before P_{2m}. □

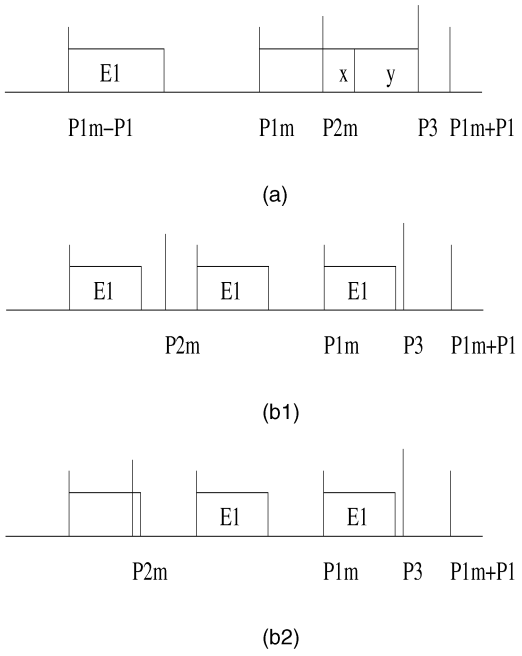


Fig. 2. Execution assignment of three-task set.

If we look at the critical-instant, the processor is busy from 0 to P_3. E_1 with the highest priority is always scheduled immediately upon request, E_2 is immediately executed in the time space left, and E_3 takes the rest of the time space.

We shall look at the execution assignment of T_1 and T_2 in the area $(\min(P_{1m}, P_{2m}), P_3)$ so that none of them will finish after P_3 and try to find the smallest utilization factor with $E_3 = P_3 - (p_1+1)E_1 - (p_2+1)E_2$.

There are two possible cases, $P_{1m} \le P_{2m}$ and $P_{1m} > P_{2m}$.

a. $P_{1m} \le P_{2m}$. Please refer to Fig. 2a. We look at the schedule in (P_{1m}, P_3).

First, we prove that the last request of T_2 before P_{2m} should be finished before P_{1m}. Suppose it is finished after P_{1m}. There are two cases of this: If the last request of T_2 before P_{2m} requests no later than the last request of T_1 before P_{1m} is finished, then $E_1 + E_2 > P_{1m} - (P_{1m} - P_1) = P_1 > P_3 - P_{1m}$; if the last request of T_2 before P_{2m} requests later than the last request of T_1 before P_{1m} is finished, then $E_2 + E_1 \ge P_{2m} - (P_{2m} - P_2) = P_2 > P_1 > P_3 - P_{1m}$. In both cases, $E_1 + E_2 > P_3 - P_{1m}$, so the last requests of T_1 and T_2 before P_3 cannot be finished before P_3. This violates the above proven property for S. So, the last request of T_2 before P_{2m} should be finished before P_{1m} and the area (P_{1m}, P_{2m}) should be all assigned to T_1.

In area (P_{2m}, P_3), suppose x is assigned to T_1, $P_3 - P_{2m} - x$ is assigned to T_2. We have $E_1 = (P_{2m} - P_{1m}) + x, E_2 = P_3 - P_{2m} - x$ and

$$E_3 = P_3 - (p_1+1)E_1 - (p_2+1)E_2.$$

The utilization $U = \frac{E_1}{P_1} + \frac{E_2}{P_2} + \frac{E_3}{P_3} = ax + b$, where

$$a = \frac{1}{P_1} - \frac{1}{P_2} - \frac{p_1 - p_2}{P_3} = \frac{p_1}{P_{1m}} - \frac{p_2}{P_{2m}} - \frac{p_1 - p_2}{P_3}$$
$$= \left(\frac{p_1 - p_2}{P_{1m}} - \frac{p_1 - p_2}{P_3}\right) + \left(\frac{p_2}{P_{1m}} - \frac{p_2}{P_{2m}}\right) > 0,$$
$$b = \frac{P_{2m} - P_{1m}}{P_1} + \frac{P_3 - P_{2m}}{P_2} + \frac{(p_1+1)P_{1m} - (p_1 - p_2)P_{2m} - p_2 P_3}{P_3}.$$

Since $a > 0$, we should assign $x = 0$.

b. $P_{1m} > P_{2m}$. Let α to be the number of T_1's requests in (P_{2m}, P_3). There are two more cases:

b1. $E_1 \le P_{2m} - (P_{1m} - \alpha P_1)$. Please refer to Fig. 2b1. $E_2 = (P_3 - P_{2m}) - \alpha E_1$. The utilization $U = \frac{E_1}{P_1} + \frac{E_2}{P_2} + \frac{E_3}{P_3} = aE_1 + b$, where

$$a = \frac{1}{P_1} - \frac{\alpha}{P_2} - \frac{(p_1+1) - \alpha(p_2+1)}{P_3},$$
$$b = \frac{P_3 - P_{2m}}{P_2} + \frac{p_2 P_{2m} + P_{2m} - p_2 P_3}{P_3}.$$

If $a \ge 0$, we should assign $E_1 = 0$; if $a < 0$, we should assign

$$E_1 = \min(P_3 - P_{1m}, P_{2m} - (P_{1m} - \alpha P_1)).$$

b2. $E_1 \ge P_{2m} - (P_{1m} - \alpha P_1)$. Please refer to Fig. 2b2. We have $P_{2m} - (P_{1m} - \alpha P_1) \le E_1 \le P_3 - P_{1m}$. The request of T_1 before P_{2m} will be finished after P_{2m}, so

$$E_2 = (P_3 - P_{2m}) - \alpha E_1 - (E_1 - (P_{2m} - (P_{1m} - \alpha P_1))).$$

The utilization $U = \frac{E_1}{P_1} + \frac{E_2}{P_2} + \frac{E_3}{P_3} = cE_1 + d$, where

$$c = \frac{1}{P_1} - \frac{\alpha+1}{P_2} - \frac{(p_1+1) - (\alpha+1)(p_2+1)}{P_3},$$
$$d = \frac{P_3 - P_{1m} + \alpha P_1}{P_2} + \frac{(p_2+1)P_{1m} - \alpha(p_2+1)P_1 - p_2 P_3}{P_3}.$$

If $c \ge 0$, we should assign

$$E_1 = P_{2m} - (P_{1m} - \alpha P_1);$$

if $c < 0$, we should assign $E_1 = P_3 - P_{1m}$.

Note we should check both cases of b to find the appropriate execution assignment.

Note also we assumed $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3$. If $\overline{U}_{\mathrm{EP}}(M_2) \ne \overline{U}_3$, the above assignments may cause T_2 unschedulable or $E_3 < 0$. Let us assign 0 to E_3 if it is calculated negative. In both situations, we can still calculate the utilization, which must not be smaller than $\overline{U}_2$.

By choosing the smaller of $\overline{U}_2$ and the utilization calculated above for $[P_1, P_2, P_3]$, we get $\overline{U}_{\mathrm{EP}}(M_2)$.

Example 3. This is a list of examples for different cases in this subsection.

- $\vec{P} = [8, 17, 18]$. This is of case a with $E_1 = 1$. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3 = 0.906$.
- $\vec{P} = [4, 15, 17]$. This is of case b. Only b1 applies. $a = 0.07 > 0$, so $E_1 = 0$. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3 = 0.898$.
- $\vec{P} = [8, 15, 17]$. This is of case b. Only b1 applies. $a = -0.005 < 0$, so $E_1 = 1$. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3 = 0.898$.
- $\vec{P} = [20, 85, 135]$. This is of case b. $\alpha = 2$. We have to check both b1 and b2.

  $$c = 0.007 > a = 0.004 > 0,$$

  so $E_1 = 0$. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_3 = 0.847$.
- $\vec{P} = [20, 70, 135]$. This is of case b. $\alpha = 3$. We have to check both b1 and b2.

  $$c = 0.0003 > 0 > a = -0.0003.$$

  So, $E_1 = P_{2m} - (P_{1m} - \alpha P_1) = 10, E_2 = 35$. This assignment causes T_2 unschedulable. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_2 = 0.929$.
- $\vec{P} = [20, 68, 135]$. This is of case b. $\alpha = 3$. We have to check both b1 and b2.

  $$0 > c = 0.001 > a = 0.002,$$

  so $E_1 = P_3 - P_{1m} = 15$. $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}_2 = 0.929$.

3.4 $\overline{U}_{\mathrm{EP}}(M_2)$ in General

The difficulty in getting exact an utilization bound for 3-task sets in the above section shows how hard it will be to find the exact utilization bound for any given $\vec{P}$. It is still open if there is a polynomial algorithm.

If we limit periods and execution times to be integers, we could derive the exact bound by exhaustively testing all possible execution assignments in the critical-instant. We shall give such an algorithm in the next section.

4 Bound Algorithms

In this section, we shall look at the properties of $\vec{P}$ and try to derive algorithms that yield utilization bounds less than $\overline{U}_{\mathrm{EP}}(M_2)$ yet better than the known bounds. In the mean time, we give another proof of Theorem 2.

Definition 6. *For a task set $S = \{T_1, T_2, \ldots, T_m\}$ with $P_1 < P_2 < \ldots < P_m$, let:*

$$P_m = p_i P_i + r_i,\ 0 \le r_i < P_i$$
$$e_i = \frac{P_i - r_i}{P_i}$$
$$\alpha_{ij} = \textit{number of requests of } T_j \textit{ between } p_i P_i \textit{ and } P_m \textit{ when } p_i P_i \le p_j P_j.$$

4.1 Reduction of $\vec{P}$ to $\vec{P}'$ where $P'_n < 2P'_1$

Theorem 6. *Given an array $\vec{P}$, let $\vec{P}' = [P'_1, P'_2, \ldots, P'_m]$ with $P'_i = p_i P_i \le P_m$ for $1 \le i < n$ and $P'_m = P_m$. Then, the minimum utilization factor of all extreme task sets satisfying $\vec{P}$ is no smaller than that of $\vec{P}'$.*

This is a similar statement to Theorem 5 in [16] and the proof idea is the same.

With the above results, we now introduce our first bound calculation algorithm:

Algorithm 1

input: **TaskPeriod**$[n]$ *in nondecreasing order*
output: utilization bound

```
var
   NewPeriod: array[1...n] of integer;

begin
   U := 1;
   for i := 2 to n do
      begin
       for j := 1 to i do
         NewPeriod[j] := TaskPeriod[j] × ⌊TaskPeriod[i] / TaskPeriod[j]⌋;
       U' := Σ_{j=1}^{i-1} (NewPeriod[j+1] − NewPeriod[j]) / NewPeriod[j]
             + (2NewPeriod[1] − NewPeriod[i]) / NewPeriod[i];
       if U > U' then U := U';
      end
   return(U);
end
```

The correctness of Algorithm 1 follows from Theorem 3, Theorem 4, and Theorem 6. Its complexity is $O(n^2)$. Note Algorithm 1 is not concerned with the task execution times as we know the exact bound of the new period vector is no larger than that of the original period vector, hence the exact bound of the new vector is a bound of the original vector.

Example 4. The utilization bound calculated according to Algorithm 1 is 0.7833 for $\vec{P} = [2, 3, 5, 6, 7, 35]$ in Example 1.

4.2 A Better Algorithm than the Harmonic Chain Method

Consider a period array $\vec{P} = [P_1, P_2, \ldots, P_n]$. According to Theorem 3, there is a smallest m, $1 \le m \le n$, $\vec{P}^m = [P_1, P_2, \ldots, P_m]$, and $\overline{U}_{\mathrm{EP}}(M_2)$ of $\vec{P}$ equals $\overline{U}_{\mathrm{EP}}(M_2)$ of $\vec{P}^m$. We define $\overline{U}_{\mathrm{EP}}(M_2)$ of $\vec{P}^m$ to be $\overline{U}^m_{\mathrm{EP}}(M_2)$.

Lemma 3. *Let $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}^m_{\mathrm{EP}}(M_2)$. If P_i divides P_j, $1 \le i < j \le m$, then $\overline{U}_{\mathrm{EP}}(M_2)$ of $\vec{P}$ equals $\overline{U}_{\mathrm{EP}}(M_2)$ of $\vec{P}' = [P_1, P_2, \ldots, P_{i-1}, P_{i+1}, \ldots, P_m]$.*

Proof. Since the extreme task set of $\vec{P}'$ is a special case of $\vec{P}^m$ with $E_i = 0$, we only need to prove that the minimum utilization factor of all extreme task sets of $\vec{P}^m$ is no less than that of $\vec{P}'$.

For any extreme task set S of $\vec{P}^m$, there is a corresponding task set S' of $\vec{P}'$ with the same utilization factor by reassigning all the execution times of E_i to the task T_j, i.e., adding $E_i \times \frac{P_j}{P_i}$ to E_j. Let's look at the critical instant of T_m in both task sets. At any time, the outstanding execution of S' is no less than that of S. Since the processor does not idle for S, so it does not for S'. This implies that, for S' increasing, E_m will cause P_m to miss deadline. So, S' is either extreme or unschedulable. If S has the minimum utilization factor, S' should

be schedulable; otherwise, we can derive from S' a *critical* task set by reducing the unschedulable execution of the unschedulable tasks. This critical task set has a smaller utilization factor. This violates the assumption that $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}^m_{\mathrm{EP}}(M_2)$. So, S' is an extreme taks set. If S of $\vec{P}^m$ has the minimum utilization factor, we have S' of $\vec{P}'$ with the same utilization factor. That means the minimum utilization factor of $\vec{P}^m$ is no smaller than that of $\vec{P}'$. □

Lemma 3 suggests that the addition of a new task whose period is a multiple of or divides an existing period is less likely to result in a smaller utilization bound for the new task set. Note that, in Lemma 3, we can only delete the period P_i, which is the divisor. Assigning execution time of P_j to that of P_i will not necessarily result in an extreme task set.

Example 5. Consider $\vec{P} = [2, 3, 6]$ and $\vec{P}' = [6]$, which is reduced from $\vec{P}$ by removing periods that divide other periods. Since the minimum utilization factor of all extreme task sets of $\vec{P}'$ is 1, we know that $\overline{U}_{\mathrm{EP}}(M_2) = 0.83333\cdots$ cannot be the minimum of all extreme task sets of $\vec{P}$. Instead, it is the minimum of all extreme task sets of $\vec{P}'' = [2, 3]$ with $E_1 = E_2 = 1$ (Theorem 3).

We can now give another proof of Theorem 2.

Proof. Look at any period array $\vec{P}$ whose elements are selected from K harmonic chains. According to Theorem 3, $\overline{U}_{\mathrm{EP}}(M_2) = \min_{i=1}^{n} \overline{U}_i$, $1 \le i \le n$. For each $\vec{P}^i$, we remove any period that divides another period in $\vec{P}^i$. Let the resulting array be $\vec{P}'^i$ and the minimum utilization factor of all extreme task sets of $\vec{P}'^i$ be $\overline{U}'_i$. We have $\overline{U}_{\mathrm{EP}}(M_2) \le \overline{U}'_i$. According to Lemma 3, $\overline{U}_{\mathrm{EP}}(M_2) = \min_{i=1}^{n} \overline{U}_i = \min_{i=1}^{n} \overline{U}'_i$.

By construction, no two tasks from the same harmonic chain can both be in S'_i since one will be removed because of the other. So, $\vec{P}'^i$ has at most K elements. By applying Theorem 1, we have $U'_i \ge K(2^{\frac{1}{K}} - 1)$. So, we have:

$$\begin{aligned}\overline{U}_{\mathrm{EP}}(M_2) &= \min_{i=1}^{n} \overline{U}_i \\ &= \min_{i=1}^{n} \overline{U}'_i \\ &\ge \min_{i=1}^{n} (K(2^{\frac{1}{K}} - 1)) \\ &= K(2^{\frac{1}{K}} - 1).\end{aligned}$$

□

Theorem 2 is a major result of [12] which improves the bound of [16]. The alternative proof above also suggests a more efficient way to compute the harmonic-chain bound than the graph-theoretic method in [12].

Algorithm 2

input: **TaskPeriod**[n] *in nondecreasing order*
output: utilization bound

var
TaskAttr: $array[1 \ldots n]$ *of integer;*
{*smallest period that divides its own period*}

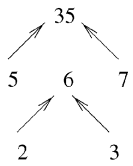


Fig. 3. The harmonic chains in $[2, 3, 5, 6, 7, 35]$.

```
begin
  for i := 1 to n - 1 do
    begin
      TaskAttr[i] := ∞;
      for j := n downto i + 1 do
        if (TaskPeriod[j] mod TaskPeriod[i] = 0)
          then TaskAttr[i] := TaskPeriod[j];
    end
  TaskAttr[n] := ∞;
  k := 0;
  for i := 1 to n do
    begin
      j := 0;
      for m := 1 to i do
        if TaskAttr[m] ≤ TaskPeriod[i] then j := j+1;
      if k <  i - j then k := i - j;
    end
  return(k · (2^(1/k) − 1));
end
```

The correctness of Algorithm 2 follows from Theorem 3 and Lemma 3. Its complexity is $O(n^2)$, which is better than that of harmonic chain ($O(n^{\frac{5}{2}})$). Even better, the bound of Algorithm 2 is actually tighter than that of the harmonic chain, as is illustrated in the following example.

Example 6. The utilization bound calculated according to Algorithm 2 is 0.7798 for $\vec{P} = [2, 3, 5, 6, 7, 35]$ in Example 1. This bound is better than the harmonic chain bound (0.7568).

Referring to Fig. 3, there are four harmonic chains in $\vec{P}$, but two chains: $(2 \to 3)$ and $(3 \to 6)$ merge at 6, which is smaller than the starting number of another chain, namely, $(7 \to 35)$. So, the k in Algorithm 2 is always smaller than 4—the number of harmonic chains in $\vec{P}$.

Algorithm 2 also suggests some intuition of how to add a new task into the task set without decreasing the utilization bound: The period of the new task should be a multiple or divisor of an existing task's period. If not, it should be chosen such that the k defined in the algorithm does not increase.

4.3 An Even Better Algorithm

We shall use the notation in Definition 6.

Lemma 4. *Let $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}^m_{\mathrm{EP}}(M_2)$. Consider P_j and P_k in $\vec{P}^m$, $p_m = p_j P_j + r_j = p_k P_k + r_k$. If $p_j P_j \le p_k P_k$ and $e_k \le \alpha_{jk} e_j$, then $\overline{U}_{\mathrm{EP}}(M_2)$ is equal to the minimum utilization factor of all extreme task sets of $\vec{P}' = [P_1, P_2, \ldots, P_{k-1}, P_{k+1}, \ldots, P_m]$.*

Proof. As in Lemma 3, we only need to show that the minimum utilization factor of all extreme task sets of $\vec{P}^m$ is no smaller than that of $\vec{P}'$.

From $e_k \le \alpha_{jk} e_j$, we get $\frac{P_k - r_k}{\alpha_{jk} P_k} \le \frac{P_j - r_j}{P_j}$. By subtracting both denominators from their enumerator, we get $\frac{P_k - r_k}{\alpha_{jk} P_k - (P_k - r_k)} \le \frac{P_j - r_j}{P_j - (P_j - r_j)}$. By definition, the denominator of the lefthand side is no greater than that of the righthand side: $\alpha_{jk} P_k - (P_k - r_k) \le r_j$. The numerator of the lefthand side must therefore be no greater than that of the righthand side: $P_k - r_k < P_j - r_j$. Summing this inequality with the last, we have, finally: $\alpha_{jk} P_k \le P_j$. Hence, the priority of T_k cannot be lower than that of T_j.

Assume the minimum is attained at $E_i, 1 \le i \le m$ and $E_k > 0$. We know the processor is always busy in $(0, P_m)$. Next. we derive a corresponding execution time assignment for $\vec{P}'$:

$$\begin{aligned} E'_l &= E_l, \text{if } l \neq j \text{ or } k \text{ or } m \\ E'_j &= E_j + \alpha_{jk} E_k \\ E'_m &= E_m - (p_j + 1)\alpha_{jk} E_k + (p_k + 1) E_k. \end{aligned}$$

In the last part from $p_j P_j$ to P_m, we subtract the same amount from E_k as we increase E_j. Since $\alpha_{jk} P_k \le P_j$, we know for sure that starting from $p_j P_j$ backward, in each of T_j's period, the increase of E_j can be scheduled in the space left by E_k. We also have $E'_m > E_m$. The increase of E_m is scheduled in the extra space left by E_k. The increase of E_m is large enough that there is no idle time from 0 to P_m. In the new schedule, the processor is still busy in $(0, P_m)$. So, the new task set either extremely utilizes the processor or is unschedulable. It cannot be unschedulable for the minimum utilization case since, otherwise, we can derive from the unschedulable task set a critical task set with a new utilization factor, which is less than $\overline{U}_{\mathrm{EP}}(M_2)$ from the following calculation. This violates the assumption that $\overline{U}_{\mathrm{EP}}(M_2)$ is equal to the minimum. Now, let us compare their utilization factors:

$$\begin{aligned} U' &= U + \alpha_{jk} \times \frac{E_k}{P_j} - \frac{E_k}{P_k} + \\ &\quad \frac{-(p_j + 1)\alpha_{jk} E_k + (p_k + 1) E_k}{P_n} \\ &= U + E_k \times \frac{e_k - \alpha_{jk} e_j}{P_n} \\ &\le U \\ &= \overline{U}_{\mathrm{EP}}(M_2). \end{aligned}$$

E_k should be 0 in order to make $\overline{U}_{\mathrm{EP}}(M_2)$ the minimum. So, the minimum utilization factor of all extreme task sets of $\vec{P}$ is equal to that of $\vec{P}'$. □

Definition 7. *Given a period array $\vec{P}$, suppose we delete from it all periods according to Lemma 3 and Lemma 4. We call the resulting period array: $\vec{P}_{\mathrm{reduced}}$ the reduced array of $\vec{P}$.*

From Theorem 3 and Lemma 4, we have the following theorem:

Theorem 7. *Let $\vec{P} = [P_1, P_2, \ldots, P_n]$.*

$$\overline{U}_{\mathrm{EP}}(M_2) = \min_{i=1}^{n} \overline{U}'_i, \ 1 \le i \le n,$$

where U'_i is the minimum utilization factor of all extreme task sets of the reduced array of $\vec{P}^i = [P_1, P_2, \ldots, P_i]$.

Algorithm 3

input: **TaskPeriod**[n] *in nondecreasing order*
output: utilization bound

```
var
   TaskPeriod1: array[1...n] of integer;
     {reduced task pattern}
   TaskPeriod2: array[1...n] of integer;
     {further reduced task pattern}

begin
   U := 1;
   for i := 1 to n do
   begin
     TaskPeriod1 := reduced task pattern of TaskPeriod;
     TaskPeriod2 := task pattern from TaskPeriod1 with
                    P := ⌊Pi/P⌋ × P for each element P;
     U' := utilization factor calculated from TaskPeriod2
           using Theorem 4;
   if U > U' then U := U';
end
   return(U);
end
```

The correctness of Algorithm 3 follows from Theorem 4, Theorem 6, and Theorem 7. Its time complexity is $O(n^3)$, but the resulting bound is better than that of Algorithm 2.

Example 7. The utilization bound calculated according to Algorithm 3 is 0.7833 for the period array $\vec{P} = [2, 3, 5, 6, 7, 35]$ of Example 1. This bound is better than the one from Algorithm 2 (0.7798).

4.4 Algorithm of Exact Bound

Before we present the last algorithm, we give two corollaries. We shall again use the notation in Definition 6.

Corollary 5. *Let $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}^m_{\mathrm{EP}}(M_2)$. In the bound case, E_m should be completed before the smallest $p_i P_i, 1 \le i < m$ and exactly on kT_i for some integer k and i.*

Proof. Suppose some δ of E_m completed after the smallest $p_i T_i$, we can derive a new task set S' with $P'_j = P_j, 1 \le j \le m$ and

$$\begin{aligned} E'_j &= E_j, \text{ if } j \neq i \text{ or } j \neq m \\ E'_i &= E_i + \frac{\epsilon}{p_i + 1} \\ E'_m &= E_m - \epsilon \end{aligned}$$

with $0 < \epsilon < \delta$. S' still extremely utilizes the processor or is unschedulable since increasing the execution time of any task will result in T'_m unschedulable. The new utilization factor:

$$\begin{aligned} U' &= U + \frac{\epsilon/(p_i+1)}{P_i} - \frac{\epsilon}{P_m} \\ &= U + \frac{\epsilon(r_i - P_i)}{(p_i+1)P_iP_m} \\ &\le U \\ &= \overline{U}^m_{\mathrm{EP}}(M_2). \end{aligned}$$

This is not possible since $\overline{U}^m_{\mathrm{EP}}(M_2)$ is supposed to be the minimum. Our assumption is thus incorrect. Also, E_m should be completed exactly on kT_i for some integer k and i; otherwise, the processor will idle between its completion time and the next nearest request of other tasks. □

Corollary 6. *Let $\overline{U}_{\mathrm{EP}}(M_2) = \overline{U}^m_{\mathrm{EP}}(M_2)$. In the bound case, for all $i, 1 \le i < m$, request of T_i at p_iP_i should be finished before P_m.*

Proof. Suppose some δ of E_i completed after P_m, we can derive a new task set S' with $P'_j = P_j, 1 \le j \le m$ and

$$\begin{aligned} E'_j &= E_j, \text{ if } j \ne i \text{ or } j \ne m \\ E'_i &= E_i - \frac{\epsilon}{p_i} \\ E'_m &= E_m + \epsilon \end{aligned}$$

with $0 < \epsilon < \delta$. S' still extremely utilizes the processor since all tasks are still schedulable and increasing the execution time of any task will result in T'_m unschedulable. The new utilization factor:

$$\begin{aligned} U' &= U - \frac{\epsilon/p_i}{P_i} + \frac{\epsilon}{P_m} \\ &= U - \frac{\epsilon r_i}{p_iP_iP_m} \\ &\le U \\ &= \overline{U}^m_{\mathrm{EP}}(M_2). \end{aligned}$$

This is not possible since $\overline{U}^m_{\mathrm{EP}}(M_2)$ is supposed to be the minimum. Our assumption is thus incorrect. □

If we assume that all periods and execution times are integer, we have the following algorithm:

Algorithm 4

input: **TaskPeriod**[n] *in nondecreasing order*
output: exact utilization bound

```
var
   r: array[1...n] of integer;
      {remains from the biggest P}

begin
   U := 1;
   for i := 1 to n do
   begin
     r:= the remains after dividing P_i;
     for each task T_j, assign execution time from 0 to r[j]
     begin
       if a combination extremely utilizes the processor
       begin
         U' := utilization factor for this case;
         if U > U' then U := U';
       end
     end
   end
   return(U);
end
```

The complexity of Algorithm 4 is exponential. This limited the size of the test we have in the next section, where we have to calculate the exact bound for comparison.

4.5 Remarks

In summary, the harmonic chain bound is better than the Liu and Layland bound. The bound from Algorithm 2 is better than the harmonic chain bound. The bound from Algorithm 1 is better than the Liu and Layland bound. It is also better than the harmonic chain bound for the array $\vec{P} = [2, 3, 5, 6, 7, 35]$ of Example 1. But, for the array $\vec{P} = [2, 4, 7]$, the harmonic chain bound (0.8284) is better than the bound from Algorithm 1 (0.8095) and both are still better than the Liu and Layland bound (0.7798). Algorithm 2 is better than the harmonic chain method; Algorithm 3 is better than Algorithm 1 and Algorithm 2. All of the bounds are smaller than that of Algorithm 4, which gives the exact bound, but is exponentially complex.

We should point out that the bound calculation algorithms are not limited to the ones we give here. We can derive different algorithms by combining the results in this paper in different ways. For example, we could combine Algorithm 1 and Algorithm 2 together. A better bound could also be derived if we could further remove periods from the $\vec{P}$ or further exploiting the period relationships. For example, [3] showed that, if all periods in a task set have values that are close to each other, a better bound than the Liu and Layland bound can be achieved.

5 Comparison of Different Bounds

The previous section gives several bound algorithms. We proved their relative tightness. In this section, we run these algorithms on randomly generated period arrays. We want to see how close to 1 the exact bound is, how close to the exact bound these bounds are, and the relative tightness among these algorithms.

Since Algorithm 4 is exponential, we are unable to test with a large number of tasks per set or big periods. The maximum task period is limited to 100. We tested with different task set size n. For each n, 100 different $\vec{P}$s are generated and their exact utilization bound are calculated. We take the average of the samples for each n. Fig. 4 shows these averages and the Liu and Layland bound of different n. We could see the exact bound is sufficiently larger than the Liu and Layland bound. Also, the Liu and Layland bound gets closer to the exact bound as n increases. One of the reasons could be because we average 100 different exact bounds, which smooths out certain bigger exact bounds of particular period arrays. Also, as n increases, the period relationship becomes more complex, the benefit of which is that the exact bounds hence decrease.

We also run the different bound algorithms on the samples. In each of the $\vec{P}$, the Liu and Layland bound is no more than the harmonic bound, which is no more than that

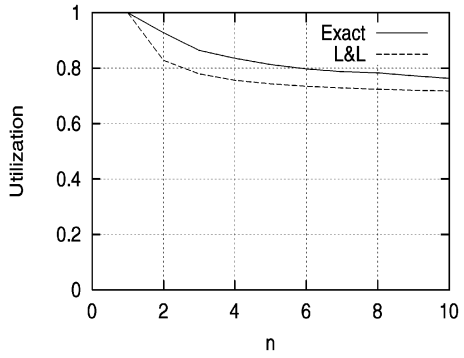


Fig. 4. The exact utilization bounds.

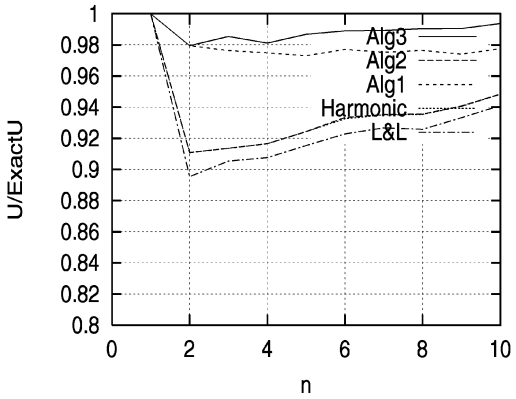


Fig. 5. Different utilization bounds.

calculated by Algorithm 2, which is, in turn, no more than that calculated by Algorithm 3, which is, further, no more than the exact bound. The bound calculated by Algorithm 1 is no less than the Liu and Layland bound and no more than that calculated by Algorithm 3. To compare their tightness, we calculate their rate to the exact bound. Again, the averages of these rates are recorded. Fig. 5 shows the averages of different algorithms for different n. The lines of harmonic and Algorithm 2 overlap each other, with that of Algorithm 2 a little bigger for some n.

When $n = 1$, all bounds equal 1. For $n > 1$, all bounds are between the Liu and Layland bound and the exact bound; they converge as n increases, like in Fig. 4. However, the bound averages do not smoothly converge with n. This could be because of the limited task sizes and the limited samples. In general, we expect the bounds to decrease with n. One interesting finding is that Algorithm 1 gives a relative better result with regard to its complexity.

6 Conclusion

There are many real-time applications where a quick online scheduling decision is necessary. The exact schedulability tests are known to be computationally difficult. Utilization bound is a good online test method where a task set is guaranteed schedulability if its utilization is no more than a bound.

In this paper, we revisited the utilization bounds for (E, P)-type tasks. We sharpen existing bounds by making the distinction between extreme and critical task sets and by making explicit the dependency of utilization bounds on the information about the task parameters. The Liu and Layland bound is a function of n—the number of tasks in a set; the harmonic chain bound is a function of K—the number of harmonic chains the task periods are from; we look at the bound given the vector of task periods. There is still no known polynomial algorithm to calculate the exact utilization bound given the task periods. We give in this paper the exact bound for some special cases.

Unable to derive the exact bound easily, we give bound algorithms that take as input the period array of an (E, P)-type task set and outperform previous bound algorithms. We also give a simpler proof for the harmonic chain bound. The algorithms are tested and compared with randomly generated task periods. The test confirms the correctness of the algorithms and the results we proved in the paper.

In many applications, the task periods are predetermined by the physical environment. For them, the algorithms provided in this paper give a faster schedulability test than critical-instant check and a better bound than previous ones.

Acknowledgments

The work of Deji Chen and Aloysius K. Mok was supported by a grant from the US Office of Naval Research under grant number N00014-98-1-0704. The work of Rei-Wei Kuo was supported in part by a research grant from the National Science Council, Taiwan, under Grant NSC89-2213-E002-200.

References

[1] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell, and A.J. Wellings, "Fixed Priority Pre-Emptive Scheduling: A Historical Perspective," *Real-Time Systems*, vol. 8, pp. 173-198, 1995.

[2] N.C. Audsley, A. Burns, M. Richardson, K.W. Tindell, and A.J. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Eng. J.*, vol. 8, no. 5, pp. 285-292, 1993.

[3] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Trans. Computers*, vol. 44, no. 12, pp. 1429-1442, Dec. 1995.

[4] A. Burns, K. Tindell, and A. Wellings, "Effective Analysis for Engineering Real-Time Fixed Priority Schedulers," *IEEE Trans. Software Eng.*, vol. 21, no. 5, pp. 475-480, May 1995.

[5] D. Chen, "Real-Time Data Management in the Distributed Environment," PhD thesis, Univ. of Texas at Austin, 1999.

[6] D. Chen, A.K. Mok, and T.-W. Kuo, "Utilization Bound Re-Visited," *Proc. Sixth Int'l Conf. Real-Time Computing Systems and Applications*, 1999.

[7] R. Devillers and J. Goossens, "Liu and Layland's Schedulability Test Revisited," *Information Processing Letters*, vol. 73, nos. 5-6, pp. 157-161, Mar. 2000.

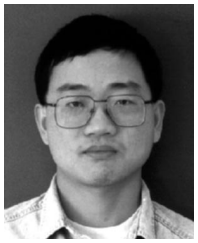
[8] C.-C. Han, "A Better Polynomial-Time Scheduleability Test for Real-Time Multiframe Tasks," *Proc. IEEE Real-Time Systems Symp.*, Dec. 1998.

[9] C.-C. Han, H.y. Tyan, "A Better Polynomial-Time Schedulability Test for Real-Time Fixed-Priority Scheduling Algorithms," *Proc. IEEE Real-Time Systems Symp.*, pp. 36-45, Dec. 1997.

[10] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer J.*, vol. 29, no. 5, pp. 390-395, Oct. 1986.

[11] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M.G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Boston: Kluwer Academic, 1993.

[12] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1991.
[13] S. Lauzac, R. Melhem, and D. Mosse, "An Efficient RMS Admission Control and Its Application to Multiprocessor Scheduling," *Proc. Int'l Parallel Processing Symp.,* pp. 511-518, 1998.
[14] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1990.
[15] J.Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation,* vol. 2, pp. 237-250, 1982.
[16] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, Jan. 1973.
[17] A.K. Mok and D. Chen, "A Multiframe Model for Real-Time Tasks," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1996.
[18] D.-W. Park, "A Generalized Utilization Bound Test for Fixed-Priority Real-Time Scheduling," PhD thesis, Texas A&M Univ.
[19] D.-W. Park, S. Natarajan, A. Kanevsky, and M.J. Kim, "A Generalized Utilization Bound Test for Fixed-Priority Real-Time Scheduling," *Proc. Second Int'l Workshop Real-Time Computing Systems and Applications,* pp. 73-77, 1995.
[20] D.-T. Peng and K.G. Shin, "A New Performance Measure for Scheduling Independent Real-Time Tasks," *J. Parallel and Distributed Computing,* vol. 19, pp. 11-26, 1993.
[21] O. Serlin, "Scheduling of Time Critical Processes," *Spring Joint Computer Conf.,* vol. 41, pp. 925-932, 1972.
[22] D. Shuzhen, X. Qiwen, and Z. Naijun, "A Formal Proof of the Rate Monotonic Scheduler," *Real-Time Computing Systems and Applications,* pp. 500-503, 1998.
[23] M. Sjodin and H. Hansson, "Improved Response-Time Analysis Calculations," *Proc. IEEE Real-Time Systems Symp.,* pp. 36-45, Dec. 1998.



Deji Chen received the BS (1989) and ME (1991) degrees in computer science and engineering from Tongji University, China. He received the MS (1993) and PhD (1999) degrees in computer science from the University of Texas at Austin. He currently works at Schlumberger Austin Technology Center. He enjoys working on real-time scheduling problems.

Aloysius K. Mok received the BS degree in electrical engineering and the MS and PhD degrees in computer science from the Massachusetts Institute of Technology. Since 1983, he has been on the faculty of the Department of Computer Sciences at the University of Texas at Austin, where he is the Quincy Lee Centennial Professor in Computer Science. Professor Mok has done extensive research on computer software systems and is internationally known for his work in real-time systems design. He is a past chairman of the Technical Committee on Real-Time Systems of the IEEE and has served on numerous research and advisory panels. His current interests include robust embedded systems, network-centric computing security, mobile systems, and real-time knowledge-based systems.

Tei-Wei Kuo received the BSE degree in computer science and information engineering from National Taiwan University in Taipei, Taiwan, in 1986. He received the MS and PhD degrees in computer sciences from the University of Texas at Austin in 1990 and 1994, respectively. He is currently an associate professor in the Department of Computer Science and Information Engineering of the National Taiwan University, Taiwan, Republic of China. He was an associate professor in the Department of Computer Science and Information Engineering of the National Chung Cheng University, Taiwan, Republic of China, from August 1994 to July 2000. His research interests include real-time databases, real-time process scheduling, real-time operating systems, and control systems. He is the program cochair of IEEE Seventh Real-Time Technology and Applications Symposium, 2001, and an associate editor of the *Journal of Real-Time Systems*, and has consulted for government and industry on problems in various real-time systems design. Dr. Kuo is a member of the IEEE Computer Society.