

# Heuristics for Minimizing Total Flow Time with Error Constraint\*

Joseph Y-T. Leung, Tommy W. Tam and C.S. Wong  
Department of Computer Science and Engineering  
University of Nebraska-Lincoln  
Lincoln, NE 68588-0115

**ABSTRACT.** We consider the problem of minimizing total flow time for the imprecise computation model introduced by Lin et al. Leung et al. have shown that the problem of finding a minimum total flow time schedule subject to the constraint that the total error is no more than a given threshold  $K$  is NP-hard, even for a single processor. In this paper we give a fast heuristic for a set of tasks with a large deadline. We show that the heuristic produces schedules with total flow time no more than  $3/2$  times the optimum solution. Examples are given showing that the ratio can asymptotically approach  $3/2$  for a single processor and  $5/4$  for multiprocessors. A second heuristic is given for a single processor and a set of tasks with different deadlines. It is shown that the worst-case performance bound of the heuristic is 2 and the bound is tight.

## 1. Introduction

In relation to the study of real-time systems Lin et al. [5-7] introduced the imprecise computation model in which a task is regarded as logically composed of two subtasks, mandatory and optional. It is required that each task completes its mandatory part, while its optional part can be left unfinished. If a task has its optional part unfinished, it incurs an error equal to the execution time of the unfinished portion. The rationale behind the imprecise computation model is to trade off the accuracy of computation for meeting the deadlines of real-time tasks. The imprecise computation

---

\*Research supported in part by the ONR grant N00014-87-K-0833 and in part by a grant from Texas Instruments, Inc.

model differs from the classical model in one important aspect. In the classical model a task must receive a processing time equal to its execution time, while in the imprecise computation model it can receive any amount no less than the execution time of its mandatory part and no more than the sum of the execution times of its mandatory and optional parts. Thus, there is an added consideration in scheduling a task, namely, deciding how much processing time should be given to a task.

The Concord System is being developed at the University of Illinois to support imprecise computations [5]. Scheduling algorithms have been proposed to minimize the total error for periodic tasks [2,7]. Recently, Shih et al. [10] studied the problem of preemptively scheduling a set of  $n$  tasks with rational ready times, deadlines and processing times on a multiprocessor system so as to minimize the total error. An  $O(n^2 \log n)$ -time algorithm is given to solve this problem. More recently, Shih et al. [9] gave a faster algorithm for a single processor. Chong and Zhao [1] gave queuing results on task scheduling to optimally trade off between the average response time and the result quality on a uniprocessor system. Leung et al. [4] have studied the problem of minimizing total flow time subject to the constraint that the total error is no more than a given threshold  $K$ . They showed that the problem is NP-hard for a set of tasks with identical ready times and a large deadline, even for a single processor [4]. For a single processor, they gave pseudo-polynomial time and polynomial time algorithms for various special cases of the problem [4].

In this paper we continue the work of [4] to study the problem of minimizing total flow time subject to the constraint that the total error is no more than a given threshold  $K$ . Motivated by the computational complexity of the problem, we propose and analyze the worst-case performance of heuristics in this paper. Two heuristics with worst-case running time of  $O(n \log n)$  are given. The first one is for multiprocessors and a set of tasks with identical ready times and a large deadline. It is shown that the heuristic produces schedules with total flow time no more than  $3/2$  times the optimum value. Examples are given showing that the ratio can asymptotically approach  $3/2$  for a single processor and  $5/4$  for multiprocessors. For small number of processors (less than six), there are examples achieving ratios larger than  $5/4$  but less than  $3/2$ . The second heuristic is for a single processor and a set of tasks with identical ready times and different deadlines. It is shown that the worst-case performance bound of the heuristic is 2 and the bound is tight.

A task system  $TS = (\{T_i\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$  with a set of  $n$  independent tasks is to be scheduled on  $p \geq 1$  identical processors. For each task  $T_i$ ,  $d(T_i)$ ,  $m(T_i)$  and  $o(T_i)$  denote its deadline, execution times of its mandatory and optional parts, respectively. Throughout this paper we assume that all parameters are integers and all tasks are ready at time 0. We

use  $e(T_i)$  to denote the total execution time of  $T_i$ ; i.e.,  $e(T_i) = m(T_i) + o(T_i)$ . A schedule is an assignment of tasks to the processors such that: (1) no processor is assigned more than one task at a time and (2) no task is assigned to more than one processor at a time. A *schedule* is said to be preemptive (nonpreemptive) if the execution of tasks are (not) allowed to be interrupted. A feasible schedule is one that satisfies the constraints: (1) no task is assigned later than its deadline and (2) each task  $T_i$  is assigned at least  $m(T_i)$  but not more than  $e(T_i)$  units of time by its deadline. A task system TS is said to be *feasible* on  $p \geq 1$  processors if there is a feasible schedule for TS on  $p$  processors.

If  $S$  is a feasible schedule for TS, we use  $\alpha(S, T_i)$  to denote the amount of time  $T_i$  is executed in  $S$ , and  $f(S, T_i)$  denotes its *finishing time*. The symbol  $\sigma(S, T_i)$  denotes the amount of time the optional part of  $T_i$  is executed in  $S$ ; i.e.,  $\sigma(S, T_i) = \alpha(S, T_i) - m(T_i)$ . The *error* generated by  $T_i$  in  $S$ , denoted by  $\epsilon(S, T_i)$ , is defined to be  $\epsilon(S, T_i) = o(T_i) - \sigma(S, T_i)$ . The *total error* of  $S$ , denoted by  $ERR(S)$ , is defined to be  $ERR(S) = \sum_{i=1}^n \epsilon(S, T_i)$ , and the *total flow time* of  $S$ , denoted by  $MFT(S)$ , is defined to be  $MFT(S) = \sum_{i=1}^n f(S, T_i)$ . Given a task system TS,  $p \geq 1$  identical processors and an error threshold  $K$ , our goal is to find a feasible preemptive (nonpreemptive) schedule  $S$  such that  $ERR(S) \leq K$  and  $MFT(S)$  is minimized. Such a schedule will be called an *optimal* schedule in this paper.

As noted above, the problem of finding optimal schedules appears to be quite difficult, even when severe constraints are put on the tasks and the number of processors. This motivated our study of fast heuristics in this paper. In the next section we give an  $O(n \log n)$ -time algorithm for a set of tasks with a large deadline to be scheduled on  $p \geq 1$  identical processors. (Note that this special case can be solved by the SPT rule in the classical model [3]. Moreover, McNaughton [8] has shown that preemption cannot reduce the total flow time. Thus, an optimal preemptive schedule has the same total flow time as an optimal nonpreemptive schedule.) We show that the algorithm generates schedules with total flow time no more than  $3/2$  times the optimum value. Examples are given showing that the ratio can asymptotically approach  $3/2$  for a single processor and  $5/4$  for multiprocessors. In Section 3 we give an  $O(n \log n)$ -time algorithm for a set of tasks with different deadlines to be scheduled on a single processor. (Note that this special case can be solved by Smith's rule in the classical model [11]. Furthermore, preemption cannot reduce the total flow time.) We show that the worst-case performance bound of the algorithm is 2 and the bound is tight. Finally, we draw some conclusions in the last section.

Throughout this paper we assume that the error threshold  $K$  is less than the total execution time of the optional parts of the tasks; i.e.,  $K < \sum_{i=1}^n o(T_i)$ . Otherwise, we can simply discard the optional part of each task and the problem will be reduced to the one in the classical model,

which can be solved in polynomial time as noted above.

## 2. Tasks with A Large Deadline

In this section we study the case where a set of tasks with a large deadline is to be scheduled on  $p \geq 1$  identical processors. For simplicity, we shall denote a task system TS by  $TS = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$  in this section. In the classical model this special case can be solved by the SPT rule [3]. The SPT rule assumes that the number of tasks  $n$  is an integral multiple of  $p$ , say  $n = lp$  for some integer  $l$ ; if not, we can add enough zero-execution-time tasks to make it so. The tasks are sorted in nondecreasing order of execution times, with the first  $p$  tasks being the rank-1 tasks, the next  $p$  tasks being the rank-2 tasks and so on. The rank-1 tasks are scheduled first, one task per processor. The rank-2 tasks are scheduled as soon as the rank-1 tasks are finished, again one task per processor. This process is repeated until all tasks have been scheduled. It is easy to see that the SPT rule can be implemented to run in  $O(n \log n)$  time. The reader is referred to [3] for a discussion of the SPT rule.

Our algorithm, to be called Algorithm A, consists of two steps. In the first step, it uses a special rule (defined later) to determine the processing time received by each task. In the second step, the tasks are scheduled by the SPT rule as in the classical model. The processing time received by each task is determined as follows. First, a list of tasks is formed in nondecreasing order of total execution times; i.e.,  $T_i$  appearing before  $T_j$  in the list implies that  $e(T_i) \leq e(T_j)$ . We then scan the list, successively discarding the optional part of the tasks until the error threshold  $K$  is reached. (Note that there is an optimal schedule such that the total error is exactly  $K$ , as shown in [4].) A formal description of Algorithm A is given below.

### Algorithm A:

**Input:** A task system  $TS = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$ , an error threshold  $K$  and  $p \geq 1$  identical processors.

**Output:** A schedule  $S_a$  with  $ERR(S_a) = K$ .

### Method:

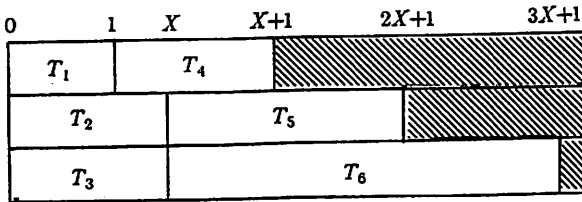
- Sort the tasks in nondecreasing order of their total execution times. Let  $L = (T_{h_1}, T_{h_2}, \dots, T_{h_n})$  be the list of tasks such that  $e(T_{h_i}) \leq e(T_{h_{i+1}})$  for each  $1 \leq i < n$ .
- Construct a classical task system  $TS' = (\{T'_i\}, \{e(T'_i)\})$  with  $n$  tasks as follows. Let  $k$  be the smallest index such that  $\sum_{i=1}^{k-1} o(T_{h_i}) < K \leq \sum_{i=1}^k o(T_{h_i})$ . Let  $e(T'_i) = m(T_{h_i})$  for each  $1 \leq i < k$ ,  $e(T'_k) = m(T_{h_k}) + (K - \sum_{i=1}^{k-1} o(T_{h_i}))$  and  $e(T'_i) = e(T_{h_i})$  for each  $k < i \leq n$ .

3. Use the SPT rule to construct a schedule  $S_a$  for  $TS'$ . □

Figure 1 shows an example task system to be scheduled on three identical processors. The schedule produced by Algorithm A is shown in Figure 1(a) and an optimal schedule is shown in Figure 1(b). It is easy to see that Algorithm A can be implemented to run in  $O(n \log n)$  time.

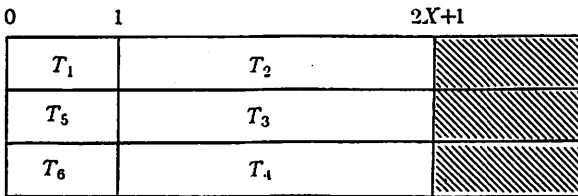
$$p = 3 \qquad K = 4X+1$$

$T_i$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$
$m(T_i)$	1	$X$	$X$	$X$	1	1
$o(T_i)$	1	$X$	$X$	$X$	$2X$	$2X$



$$MFT(S_a) = 8X+4$$

Figure 1(a). The Schedule  $S_a$ .



$$MFT(S_o) = 6X+6$$

Figure 1(b). The Schedule  $S_o$ .

**Figure 1.** Example Task System Illustrating Algorithm A.

We note that the ordering in the first step of Algorithm A must be properly chosen to ensure a reasonably good performance bound. A poor choice can lead to a schedule with total flow time arbitrarily large compared to the optimum value. For example, suppose that the tasks are sorted in nondecreasing order of the mandatory execution times. The new rule

will give an unbounded performance ratio as the following example shows. Consider the task system  $TS = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$  with the following  $n$  tasks to be scheduled on a single processor:  $(T_1, 0, nX)$  and  $(T_i, 1, X)$  for  $2 \leq i \leq n$ , where  $X$  is an arbitrary large integer. We choose  $K$  to be  $(n-1)X$ . Let  $S$  and  $S_o$  be the schedules obtained by the new rule and an optimal algorithm, respectively. Clearly, we have  $\alpha(S, T_1) = X$  and  $\alpha(S, T_i) = 1 + X$  for  $2 \leq i \leq n$ . The tasks are scheduled in  $S$  in the order  $T_1, T_2, \dots, T_n$ , with  $MFT(S) = n(n+1)X/2 + n(n-1)/2$ . However, in  $S_o$  we have  $\alpha(S_o, T_1) = nX$  and  $\alpha(S_o, T_i) = 1$  for  $2 \leq i \leq n$ . The tasks are scheduled in  $S_o$  in the order  $T_2, T_3, \dots, T_n, T_1$ , with  $MFT(S_o) = nX + (n^2 + n - 2)/2$ . Clearly,  $MFT(S)/MFT(S_o)$  approaches infinity as  $n$  and  $X$  approach infinity.

Before we prove the worst-case performance bound for Algorithm A, we will give example task systems achieving the largest performance ratios for various values of  $p$ ; they are summarized in Theorem 2.1. In the following  $S_a$  and  $S_o$  denote the schedules obtained for a given task system by Algorithm A and an optimal algorithm, respectively.

**Theorem 2.1.** *For  $p = 1, 2, 3, 4$  and  $5$ , there are task systems such that  $MFT(S_a)/MFT(S_o)$  can asymptotically approach  $3/2, 4/3, 4/3, 9/7$  and  $13/10$ , respectively. For  $p$  larger than  $5$  and even, there are task systems such that  $MFT(S_a)/MFT(S_o)$  can asymptotically approach  $5/4$ .*

**Proof:** For  $p = 1$ , consider the task system  $TS = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$  with the following 2 tasks:  $(T_1, X, X)$  and  $(T_2, 1, 2X)$ , where  $X$  is an arbitrary large integer. Let  $K$  be  $2X$ . Algorithm A discards all of the optional part of  $T_1$  and  $X$  units of  $T_2$ . An optimal algorithm discards all of the optional part of  $T_2$ . Clearly, we have  $MFT(S_a)/MFT(S_o) = (3X+1)/(2X+2)$ . Thus,  $MFT(S_a)/MFT(S_o) \rightarrow 3/2$  as  $X \rightarrow \infty$ .

For  $p = 2$ , consider the task system  $TS$  with the following 4 tasks:  $(T_1, 1, 1)$ ,  $(T_2, X, X)$ ,  $(T_3, X, X)$  and  $(T_4, 1, 2X)$ . Let  $K = 3X + 1$ . Algorithm A discards all of the optional parts of  $T_1$  to  $T_3$  and  $X$  units of  $T_4$ . An optimal algorithm discards all of the optional parts of  $T_1, T_2$  and  $T_4$ . We have  $MFT(S_a)/MFT(S_o) = (4X + 3)/(3X + 4)$ . Thus,  $MFT(S_a)/MFT(S_o) \rightarrow 4/3$  as  $X \rightarrow \infty$ .

For  $p = 3$ , consider the task system  $TS$  shown in Figure 1. It is clear that  $MFT(S_a)/MFT(S_o) \rightarrow 4/3$  as  $X \rightarrow \infty$ .

For  $p = 4$ , consider the task system  $TS$  with the following 8 tasks:  $(T_i, 1, 1)$  for  $i = 1$  and  $2$ ,  $(T_i, X, X)$  for  $3 \leq i \leq 6$ , and  $(T_i, 1, 2X)$  for  $i = 7$  and  $8$ . Let  $K = 5X + 2$ . Algorithm A discards all of the optional parts of  $T_1$  to  $T_6$  and  $X$  units of  $T_7$ . An optimal algorithm discards all of the optional parts of  $T_1$  to  $T_3$  and  $T_7$  to  $T_8$ . We have  $MFT(S_a)/MFT(S_o) = (9X + 6)/(7X + 8)$ . Thus,  $MFT(S_a)/MFT(S_o) \rightarrow 9/7$  as  $X \rightarrow \infty$ .

For  $p = 5$ , consider the task system with the following 10 tasks:  $(T_i, 1, 1)$

for  $i = 1$  and  $2$ ,  $(T_i, X, X)$  for  $3 \leq i \leq 7$ , and  $(T_i, 1, 2X)$  for  $8 \leq i \leq 10$ . Let  $K = 6X + 2$ . Algorithm A discards all of the optional parts of  $T_1$  to  $T_7$  and  $X$  units of  $T_8$ . An optimal algorithm discards all of the optional parts of  $T_1$  to  $T_2$  and  $T_8$  to  $T_{10}$ . We have  $\text{MFT}(S_a)/\text{MFT}(S_o) = (13X + 7)/(10X + 10)$ . Thus,  $\text{MFT}(S_a)/\text{MFT}(S_o) \rightarrow 13/10$  as  $X \rightarrow \infty$ .

For  $p > 5$  and even, consider the task system with the following  $2p$  tasks:  $(T_i, 1, 1)$  for  $1 \leq i \leq p/2$ ,  $(T_i, X, X)$  for  $p/2 < i \leq 3p/2$ , and  $(T_i, 1, 2X)$  for  $3p/2 < i \leq 2p$ . Let  $K = pX + p/2$ . Algorithm A discards all of the optional parts of  $T_1$  to  $T_{3p/2}$ . An optimal algorithm discards all of the optional parts of  $T_1$  to  $T_{p/2}$  and  $T_{3p/2+1}$  to  $T_{2p}$ . We have  $\text{MFT}(S_a)/\text{MFT}(S_o) = (5pX + 3p)/(4pX + 4p)$ . Thus,  $\text{MFT}(S_a)/\text{MFT}(S_o) \rightarrow 5/4$  as  $X \rightarrow \infty$ .

We note that all of the above examples can be generalized to have an arbitrarily large number of tasks.  $\square$

Before we prove the worst-case performance bound, we need to introduce some notations and definitions. Let  $\text{TS} = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$  be a set of  $n$  independent tasks to be scheduled on  $p \geq 1$  identical processors. We assume that  $n = lp$  for some positive integer  $l$ ; otherwise, we can add enough zero-execution-time tasks to make it so. Let  $T_{g_1}, T_{g_2}, \dots, T_{g_n}$  be an ordering of the tasks in  $\text{TS}$  such that  $m(T_{g_i}) \leq m(T_{g_{i+1}})$  for each  $1 \leq i \leq n - 1$  and let  $T_{h_1}, T_{h_2}, \dots, T_{h_n}$  be an ordering such that  $e(T_{h_i}) \leq e(T_{h_{i+1}})$  for each  $1 \leq i \leq n - 1$ . Let  $S_o$  and  $S_a$  be an optimal schedule and the schedule produced by Algorithm A, respectively, for  $\text{TS}$  on  $p$  processors. Note that Algorithm A differs from an optimal algorithm only in the amounts of processing time assigned to the tasks, as they both schedule tasks by the SPT rule. Since both Algorithm A and an optimal algorithm schedule tasks by the SPT rule,  $S_a$  and  $S_o$  must have the following three properties: (1) it is a nonpreemptive schedule, (2) there are exactly  $l$  tasks scheduled on each processor and (3) the processing time of any task scheduled in the  $i$ th position (from the beginning) on any processor is no larger than the processing time of any task scheduled in the  $j$ th position (from the beginning) on any processor, where  $1 \leq i < j \leq l$ .

Let  $T_{x_{(i-1)p+j}}$  and  $T_{y_{(i-1)p+j}}$  be the tasks scheduled in the  $i$ th position (from the beginning) on the  $j$ th processor in  $S_a$  and  $S_o$ , respectively, where  $1 \leq i \leq l$  and  $1 \leq j \leq p$ . Let  $\rho_i(S_a) = \{T_{x_{(i-1)p+1}}, T_{x_{(i-1)p+2}}, \dots, T_{x_{ip}}\}$  and  $\rho_i(S_o) = \{T_{y_{(i-1)p+1}}, T_{y_{(i-1)p+2}}, \dots, T_{y_{ip}}\}$  be the rank- $i$  tasks in  $S_a$  and  $S_o$ , respectively,  $1 \leq i \leq l$ . We define  $\Pi_i(S_a)$  and  $\Pi_i(S_o)$  to be the total processing time of all the tasks in  $\rho_i(S_a)$  and  $\rho_i(S_o)$ , respectively. That is,  $\Pi_i(S_a) = \sum_{j=1}^p \alpha(S_a, T_{x_{(i-1)p+j}})$  and  $\Pi_i(S_o) = \sum_{j=1}^p \alpha(S_o, T_{y_{(i-1)p+j}})$ . The symbols  $\Phi_i(S_a)$  and  $\Phi_i(S_o)$  denote the total finishing times of all the tasks in  $\rho_i(S_a)$  and  $\rho_i(S_o)$ , respectively. That is,  $\Phi_i(S_a) = \sum_{j=1}^p f(S_a, T_{x_{(j-1)p+j}})$  and  $\Phi_i(S_o) = \sum_{j=1}^p f(S_o, T_{y_{(i-1)p+j}})$ . Clearly,  $\text{MFT}(S_a) = \sum_{i=1}^l \Phi_i(S_a)$  and  $\text{MFT}(S_o) = \sum_{i=1}^l \Phi_i(S_o)$ . Similarly,  $\text{MFT}(S_a) = \sum_{j=1}^p \Pi_j(S_a)$ . Similarly,  $\text{MFT}(S_o) = \sum_{j=1}^p \Pi_j(S_o)$ .

and  $\Phi_i(S_o) = \sum_{j=1}^i \Pi_j(S_o)$ . Finally, let  $k$  be the index obtained in Step (2) of Algorithm A and  $\kappa$  be such that  $(\kappa - 1)p + 1 \leq k \leq \kappa p$ .

In the remainder of this section we will show that the worst-case performance bound of Algorithm A is  $3/2$ . The next five lemmas are instrumental in proving the result.

**Lemma 2.1.**  $\sum_{i=\kappa}^l \Phi_i(S_a) \leq \sum_{i=\kappa}^l \Phi_i(S_o)$ .

**Proof:** By the nature of Algorithm A, we have  $\rho_i(S_a) = \{T_{h_{(i-1)p+1}}, T_{h_{(i-1)p+2}}, \dots, T_{h_{ip}}\}$  for each  $\kappa + 1 \leq i \leq l$ . Furthermore,  $\alpha(S_a, T_j) = e(T_j)$  for each  $T_j \in \rho_i(S_a)$ ,  $\kappa + 1 \leq i \leq l$ . Thus, we have

$$\Pi_i(S_a) \geq \Pi_i(S_o) \text{ for each } \kappa + 1 \leq i \leq l. \quad (2.1)$$

Since  $\sum_{i=1}^l \Pi_i(S_a) = \sum_{i=1}^n e(T_i) - K = \sum_{i=1}^l \Pi_i(S_o)$ , we have  $\Phi_l(S_a) = \Phi_l(S_o)$ . Furthermore, for each  $\kappa \leq i \leq l - 1$ ,  $\Phi_i(S_a) = \sum_{j=1}^i \Pi_j(S_a) = \sum_{j=1}^i \Pi_j(S_o) - \sum_{j=i+1}^l \Pi_j(S_a)$  and  $\Phi_i(S_o) = \sum_{j=1}^i \Pi_j(S_o) = \sum_{j=1}^i \Pi_j(S_o) - \sum_{j=i+1}^l \Pi_j(S_o)$ . Thus, we have from (2.1),  $\Phi_i(S_a) \leq \Phi_i(S_o)$  for each  $\kappa \leq i \leq l - 1$ . The lemma follows immediately.  $\square$

**Lemma 2.2.** If  $\kappa - 1 \leq l/2$ , then  $MFT(S_a) \leq (3/2)MFT(S_o)$ .

**Proof:** Let us consider the tasks scheduled on the  $j$ th processor in  $S_a$ ,  $1 \leq j \leq p$ . Since  $\alpha(S_a, T_{x_j}) \leq \alpha(S_a, T_{x_{p+j}}) \leq \dots \leq \alpha(S_a, T_{x_{(\kappa-2)p+j}}) \leq \alpha(S_a, T_{x_{(\kappa-1)p+j}}) \leq \dots \leq \alpha(S_a, T_{x_{(l-1)p+j}})$  and  $\kappa - 1 \leq l/2$ , we have  $f(S_a, T_{x_{(i-1)p+j}}) \leq (1/2)f(S_a, T_{x_{(\kappa-2+i)p+j}})$  for each  $1 \leq i \leq \kappa - 1$ . Therefore, we have

$$\begin{aligned} \sum_{i=1}^{\kappa-1} \Phi_i(S_a) &\leq (1/2) \sum_{i=1}^{\kappa-1} \Phi_{\kappa-1+i}(S_a) \\ &\leq (1/2) \sum_{i=\kappa}^l \Phi_i(S_a). \end{aligned}$$

Now, we have

$$\begin{aligned} MFT(S_a) &= \sum_{i=1}^{\kappa-1} \Phi_i(S_a) + \sum_{i=\kappa}^l \Phi_i(S_a) \\ &\leq (3/2) \sum_{i=\kappa}^l \Phi_i(S_a) \\ &\leq (3/2) \sum_{i=\kappa}^l \Phi_i(S_o) \text{ (by Lemma 2.1)} \\ &\leq (3/2)MFT(S_o). \end{aligned} \quad \square$$



As a result of Lemma 2.2, we can concentrate on the case  $\kappa - 1 > 1/2$ . In the next three lemmas, we assume that  $\kappa - 1 > 1/2$ .

**Lemma 2.3.**  $\sum_{i=1}^{2\kappa-l-2} \Phi_i(S_a) \leq \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o)$ .

**Proof:** There are at least  $(\kappa - 1)p$  tasks without any optional part in the set  $\rho_1(S_a) \cup \rho_2(S_a) \cup \dots \cup \rho_\kappa(S_a)$ . The set  $\{T_{g(l-\kappa+1)p+1}, T_{g(l-\kappa+1)p+2}, \dots, T_{glp}\}$  contains  $(\kappa - 1)p$  tasks with the largest mandatory parts. Therefore, we have for each  $1 \leq i \leq \kappa - 1$ ,  $\Pi_i(S_a) \leq \sum_{u=1}^p m(T_{g(l-\kappa+i)p+u})$ , and hence,  $\Phi_i(S_a) \leq \sum_{j=1}^i [\sum_{u=1}^p m(T_{g(l-\kappa+j)p+u})]$ . We also have for each  $1 \leq i \leq l$ ,  $\Pi_i(S_o) \geq \sum_{u=1}^p m(T_{g(i-1)p+u})$ , and hence,  $\Phi_i(S_o) \geq \sum_{j=1}^i [\sum_{u=1}^p m(T_{g(j-1)p+u})]$ . From the above properties, we have

$$\begin{aligned} \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) &\geq \sum_{i=l-\kappa+2}^{\kappa-1} \left[ \sum_{j=1}^i \left[ \sum_{u=1}^p m(T_{g(j-1)p+u}) \right] \right] \\ &\geq \sum_{i=l-\kappa+2}^{\kappa-1} \left[ \sum_{j=l-\kappa+2}^i \left[ \sum_{u=1}^p m(T_{g(j-1)p+u}) \right] \right] \\ &= \sum_{i=1}^{2\kappa-l-2} \left[ \sum_{j=1}^i \left[ \sum_{u=1}^p m(T_{g(l-\kappa+j)p+u}) \right] \right] \\ &\geq \sum_{i=1}^{2\kappa-l-2} \Phi_i(S_a). \end{aligned}$$

□

**Lemma 2.4.**  $(l - \kappa + 1)\Phi_{2\kappa-l-2}(S_a) \leq \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + \sum_{i=\kappa}^{l-1} [(l - i)\Pi_i(S_a)]$ .

**Proof:** By definition, we have  $\Phi_i(S_o) = \sum_{j=1}^i \Pi_j(S_o)$  for each  $1 \leq i \leq l$ . Therefore, we have

$$\Phi_i(S_o) \geq \sum_{j=l-\kappa+2}^i \Pi_j(S_o) \text{ for each } l - \kappa + 2 \leq i \leq l. \quad (2.2)$$

From the proof of Lemma 2.3, we have for each  $1 \leq i \leq 2\kappa - l - 2$ ,  $\Pi_{l-\kappa+1+i}(S_o) \geq \sum_{u=1}^p m(T_{g(l-\kappa+i)p+u}) \geq \Pi_i(S_a)$ . Therefore, we have

$$\Pi_{l-\kappa+1+i}(S_o) \geq \Pi_i(S_a) \text{ for each } 1 \leq i \leq 2\kappa - l - 2. \quad (2.3)$$

Since Algorithm A uses the SPT rule to schedule tasks, we have

$$\Pi_j(S_a) \geq \Pi_i(S_a) \text{ for each } 1 \leq i \leq 2\kappa - l - 2 \text{ and } \kappa \leq j \leq l - 1. \quad (2.4)$$

Using (2.2), (2.3) and (2.4), we obtain

$$\begin{aligned}
 & \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + \sum_{i=\kappa}^{l-1} [(l-i)\Pi_i(S_o)] \\
 & \geq \sum_{i=l-\kappa+2}^{\kappa-1} \sum_{j=l-\kappa+2}^i \Pi_j(S_o) + \sum_{i=\kappa}^{l-1} [(l-i)\Pi_i(S_o)] \text{ (by (2.2))} \\
 & = \sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i)\Pi_{l-\kappa+1+i}(S_o)] + \sum_{i=\kappa}^{l-1} [(l-i)\Pi_i(S_o)] \\
 & \geq \sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i)\Pi_i(S_a)] + \sum_{i=1}^{l-\kappa} [(l-\kappa+1-i)\Pi_{2\kappa-l-1-i}(S_a)], \\
 & \quad \text{(by (2.3) and (2.4))}
 \end{aligned}$$

where  $\Pi_i(S_a) = 0$  for  $i \leq 0$ .

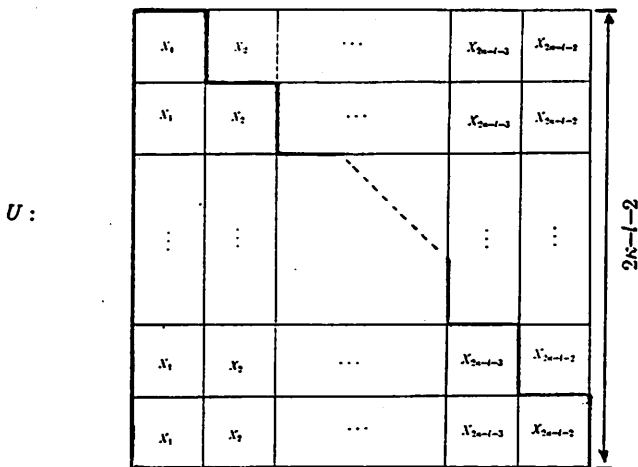


Figure 2(a). The Matrix  $U$ .

Shown in Figure 2(a) is a square matrix  $U$  with dimension  $2\kappa - l - 2$ . All rows of  $U$  contain  $X_1, X_2, \dots$ , and  $X_{2\kappa-l-2}$ , where  $X_j = \prod_j(S_a)$  for each  $1 \leq j \leq 2\kappa - l - 2$ . From Figure 2(a), we see that the sum of the entries in the lower triangle of  $U$  (including the diagonal elements) is exactly  $\sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i)\Pi_i(S_a)]$ . Shown in Figure 2(b) is a square matrix  $V$  with dimension  $l-\kappa$ . All rows of  $V$  contain  $X_{3\kappa-2l-1}, X_{3\kappa-2l}, \dots$ , and  $X_{2\kappa-l-2}$ , where  $X_j = \prod_j(S_a)$  if  $j \geq 1$  and  $X_j = 0$  if  $j \leq 0$ . From Figure 2(b), we see that the sum of the entries in the upper triangle of  $V$  (including the diagonal elements) is exactly  $\sum_{i=1}^{l-\kappa} [(l-\kappa+1-i)\Pi_{2\kappa-l-1-i}(S_a)]$ .

We claim that  $\sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i) \prod_i(S_a)] + \sum_{i=1}^{l-\kappa} [(l-\kappa+1-i) \prod_{2\kappa-l-1-i}(S_a)] \geq (l-\kappa+1) \sum_{i=1}^{2\kappa-l-2} \prod_i(S_a)$ .

$$X_i = \begin{cases} \pi_i(S_a) & \text{if } i \geq 1 \\ 0 & \text{if } i \leq 0 \end{cases}$$

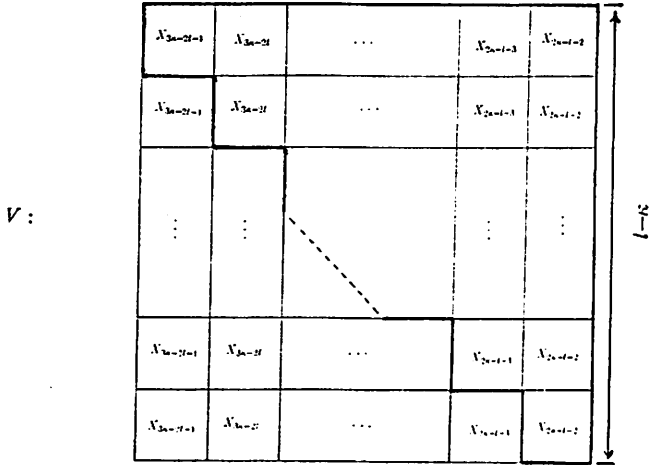


Figure 2(b). The Matrix  $V$ .

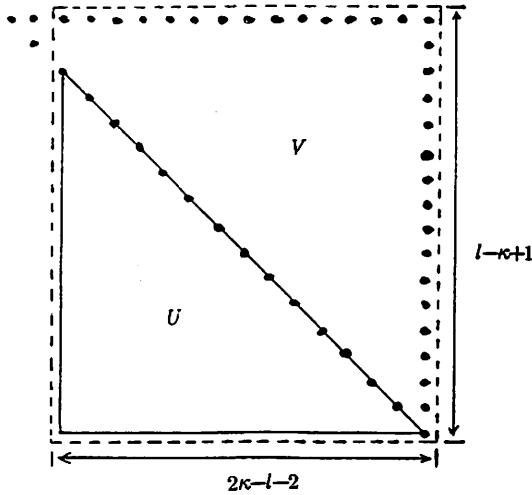


Figure 2(c). The Case  $l-\kappa \geq 2\kappa-l-2$ .

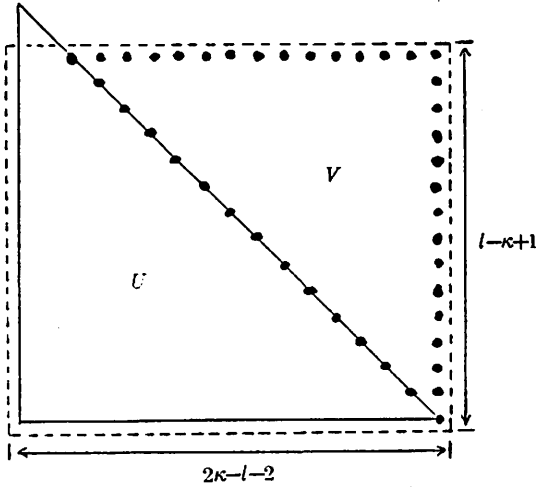


Figure 2(d). The Case  $l - \kappa < 2\kappa - l - 2$ .

There are two cases to consider, depending on whether  $l - \kappa \geq 2\kappa - l - 2$  or not. Suppose  $l - \kappa \geq 2\kappa - l - 2$ . Shown in Figure 2(c) is the upper triangle of  $V$  (the dotted triangle) put on top of the lower triangle of  $U$  (the solid triangle). The sum of the entries in the  $(l - \kappa + 1)$  by  $(2\kappa - l - 2)$  rectangle enclosed in dash lines is exactly  $(l - \kappa + 1) \sum_{i=1}^{2\kappa-l-2} \Pi_i(S_a)$ . Clearly, we have  $\sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i) \Pi_i(S_a)] + \sum_{i=1}^{l-\kappa} [(l-\kappa+1-i) \Pi_{2\kappa-l-1-i}(S_a)] \geq (l - \kappa + 1) \sum_{i=1}^{2\kappa-l-2} \Pi_i(S_a)$ . On the other hand, if  $l - \kappa < 2\kappa - l - 2$ , then consider Figure 2(d) where the upper triangle of  $V$  (the dotted triangle) is put on top of the lower triangle of  $U$  (the solid triangle). The sum of the entries in the  $(l - \kappa + 1)$  by  $(2\kappa - l - 2)$  rectangle enclosed in dashed lines is exactly  $(l - \kappa + 1) \sum_{i=1}^{2\kappa-l-2} \Pi_i(S_a)$ . Clearly,  $\sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i) \Pi_i(S_a)] + \sum_{i=1}^{l-\kappa} [(l-\kappa+1-i) \Pi_{2\kappa-l-1-i}(S_a)] \geq (l - \kappa + 1) \sum_{i=1}^{2\kappa-l-2} \Pi_i(S_a)$ . Thus, we have

$$\begin{aligned}
 & \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + \sum_{i=\kappa}^{l-1} [(l-i)\Pi_i(S_a)] \\
 & \geq \sum_{i=1}^{2\kappa-l-2} [(2\kappa-l-1-i)\Pi_i(S_a)] + \sum_{i=1}^{l-\kappa} [(l-\kappa+1-i)\Pi_{2\kappa-l-1-i}(S_a)] \\
 & \geq (l - \kappa + 1) \sum_{i=1}^{2\kappa-l-2} \Pi_i(S_a) \\
 & = (l - \kappa + 1) \Phi_{2\kappa-l-2}(S_a).
 \end{aligned}$$

□

**Lemma 2.5.**  $\sum_{i=2\kappa-l-1}^{\kappa-1} \Phi_i(S_a) \leq (1/2) \sum_{i=l-\kappa+2}^l \Phi_i(S_o)$ .


**Proof:** We have

$$\begin{aligned}
 & \sum_{i=l-\kappa+2}^l \Phi_i(S_o) \\
 &= \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + \sum_{i=\kappa}^l \Phi_i(S_o) \\
 &\geq \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + \sum_{i=\kappa}^l \Phi_i(S_a) \text{ (by Lemma 2.1)} \\
 &= \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + (l-\kappa+1)\Phi_{\kappa-1}(S_a) + \sum_{i=\kappa}^l [(l+1-i)\Pi_i(S_a)] \\
 &= \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + (l-\kappa+1) \left[ \Phi_{2\kappa-l-2}(S_a) + \sum_{i=1}^{l-\kappa+1} \Pi_{2\kappa-l-2+i}(S_a) \right] \\
 &+ \sum_{i=\kappa}^l (l-i+1)\Pi_i(S_a) \\
 &\geq 2(l-\kappa+1)\Phi_{2\kappa-l-2}(S_a) + (l-\kappa+1) \sum_{i=1}^{l-\kappa+1} \Pi_{2\kappa-l-2+i}(S_a) \\
 &+ \sum_{i=\kappa}^l \Pi_i(S_a) \text{ (by Lemma 2.4)}
 \end{aligned}$$

Since  $\prod_j(S_a) \geq \prod_i(S_a)$  for each  $1 \leq i \leq \kappa-1$  and  $\kappa \leq j \leq l$ , we have

$$\begin{aligned}
 & \sum_{i=l-\kappa+2}^l \Phi_i(S_o) \\
 &\geq 2(l-\kappa+1)\Phi_{2\kappa-l-2}(S_a) + (l-\kappa+1) \sum_{i=1}^{l-\kappa+1} \Pi_{2\kappa-l-2+i}(S_a) \\
 &+ \sum_{i=1}^{l-\kappa+1} \Pi_{2\kappa-l-2+i}(S_a) \\
 &= 2(l-\kappa+1)\Phi_{2\kappa-l-2}(S_a) + (l-\kappa+2) \sum_{i=1}^{l-\kappa+1} \Pi_{2\kappa-l-2+i}(S_a).
 \end{aligned}$$

$l - \kappa + 1$

	$X_1$	$X_2$	$\dots$	$X_{l-\kappa}$	$X_{l-\kappa+1}$
	$X_1$	$X_2$	$\dots$	$X_{l-\kappa}$	$X_{l-\kappa+1}$
	$X_1$	$X_2$	$\dots$	$X_{l-\kappa}$	$X_{l-\kappa+1}$
$W :$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	$X_1$	$X_2$	$\dots$	$X_{l-\kappa}$	$X_{l-\kappa+1}$
	$X_1$	$X_2$	$\dots$	$X_{l-\kappa}$	$X_{l-\kappa+1}$

$l - \kappa + 2$

$$X_j = \pi_{2\kappa-l-2+j}(S_a), \quad 1 \leq j \leq l - \kappa + 1$$

**Figure 3.** Illustrating the Proof of Lemma 2.5.

Shown in Figure 3 is a  $(l - \kappa + 2)$  by  $(l - \kappa + 1)$  array  $W$ . All rows of  $W$  contain  $X_1, X_2, \dots, X_{l-\kappa+1}$ , where  $X_j = \pi_{2\kappa-l-2+j}(S_a)$ ,  $1 \leq j \leq l - \kappa + 1$ . Clearly, the sum of the entries in the array  $W$  is exactly  $(l - \kappa + 2) \sum_{i=1}^{l-\kappa+1} \pi_{2\kappa-l-2+i}(S_a)$ . The sum of the entries enclosed in the solid triangle in Figure 3 is exactly  $\sum_{i=1}^{l-\kappa+1} (l - \kappa + 2 - i) \pi_{2\kappa-l-2+i}(S_a)$ . Since  $X_j \geq X_i$  whenever  $j \geq i$ , the sum of all the entries in the array  $W$  must be at least twice that in the solid triangle. Thus, we have  $(l - \kappa + 2) \sum_{i=1}^{l-\kappa+1} \pi_{2\kappa-l-2+i}(S_a) \geq 2 \sum_{i=1}^{l-\kappa+1} (l - \kappa + 2 - i) \pi_{2\kappa-l-2+i}(S_a)$ ,

and hence

$$\begin{aligned}
 & \sum_{i=l-\kappa+2}^l \Phi_i(S_o) \\
 & \geq 2(l-\kappa+1)\Phi_{2\kappa-l-2}(S_a) + 2 \sum_{i=1}^{l-\kappa+1} (l-\kappa+2-i)\Pi_{2\kappa-l-2+i}(S_a) \\
 & = 2 \sum_{i=2\kappa-l-1}^{\kappa-1} \Phi_i(S_a).
 \end{aligned}$$

□

Using the above lemmas, we can prove the main result in this section.

**Theorem 2.2.** *For any task system  $TS = (\{T_i\}, \{m(T_i)\}, \{o(T_i)\})$  to be scheduled on  $p \geq 1$  identical processors, we have  $MFT(S_a) \leq (3/2)MFT(S_o)$ .*

**Proof:** If  $\kappa - 1 \leq l/2$ , the theorem follows immediately from Lemma 2.2. Thus, we may assume that  $\kappa - 1 > l/2$ . We have

$$\begin{aligned}
 & MFT(S_a) \\
 & = \sum_{i=1}^{2\kappa-l-2} \Phi_i(S_a) + \sum_{i=2\kappa-l-1}^{\kappa-1} \Phi_i(S_a) + \sum_{i=\kappa}^l \Phi_i(S_a) \\
 & \leq \sum_{i=l-\kappa+2}^{\kappa-1} \Phi_i(S_o) + (1/2) \sum_{i=l-\kappa+2}^l \Phi_i(S_o) + \sum_{i=\kappa}^l \Phi_i(S_o) \\
 & \quad \text{(by Lemmas 2.3, 2.5, and 2.1)} \\
 & = \sum_{i=l-\kappa+2}^l \Phi_i(S_o) + (1/2) \sum_{i=l-\kappa+2}^l \Phi_i(S_o) \\
 & = (3/2) \sum_{i=l-\kappa+2}^l \Phi_i(S_o) \\
 & \leq (3/2)MFT(S_o).
 \end{aligned}$$

□

### 3. Tasks with Different Deadlines

In this section we assume that a set of tasks with different deadlines is to be scheduled on a single processor. As noted in Section 1, this special case can be solved by Smith's rule in the classical model [11]. Our proposed heuristic, Algorithm B, employs Smith's rule to schedule tasks. Smith's

rule assumes that the task system  $TS = (\{T_i\}, \{d(T_i)\}, \{e(T_i)\})$  satisfies the property that  $SL = \sum_{i=1}^n e(T_i) = \max_{i=1}^n \{d(T_i)\}$ ; if  $SL < \max\{d(T_i)\}$ , we can reduce those deadlines larger than  $SL$  to simply  $SL$ . Smith's rule works as follows: Schedule the tasks backwards, starting at the latest deadline. At each decision point, schedule last the largest task (in terms of execution time) from among those that can be scheduled last. Using a heap, Smith's rule can be implemented to run in  $O(n \log n)$  time. Note that the schedule obtained by Smith's rule has length exactly  $SL$  and it is a nonpreemptive schedule.

For the imprecise computation model, the special case studied in this section has been shown to be NP-hard [4]. Moreover, it has been shown in [4] that there is always an optimal schedule  $S$  for a task system  $TS = \{T_i\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\}$  with the following properties:  $S$  is a nonpreemptive schedule with no idle time,  $ERR(S) = K$ , and there is an index  $l$  such that the first  $l - 1$  tasks scheduled in  $S$  have no optional part, the last  $n - l$  tasks scheduled in  $S$  have full optional parts, and the  $l$ th task has some (possibly all) optional part. Thus,  $S$  must have length  $SL' = \sum_{i=1}^n e(T_i) - K$ .

In this section we give a heuristic, to be called Algorithm B, for this special case. We show that the worst-case performance bound of Algorithm B is 2 and the bound is tight. As in Algorithm A, Algorithm B also operates in two steps. In the first step, it uses a special rule (to be defined later) to determine the processing time assigned to each task. In the second step, the tasks are scheduled by Smith's rule. The processing time assigned to each task is determined as follows. We attempt to schedule the tasks by Smith's rule, starting at time  $SL'$ . The first few tasks scheduled by Smith's rule (these tasks must appear at the end of the schedule) are assigned processing times equal to their total execution times. This process is repeated until we reach a point where exactly  $(\sum_{i=1}^n o(T_i) - K)$  units of optional parts have been assigned. The unscheduled tasks are all given processing times equal to the execution times of their mandatory parts only. Shown below is a formal description of Algorithm B.

**Algorithm B:**

Input: A task system  $TS = (\{T_i\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$ , an error threshold  $K$  and a single processor.

Output: A schedule  $S_b$  with  $ERR(S_b) = K$ .

Method:

1.  $t \leftarrow \sum_{i=1}^n e(T_i) - K$ .  $\Theta \leftarrow \sum_{i=1}^n o(T_i) - K$ .  $TS' \leftarrow$  an empty task system in the classical model.  $l \leftarrow 1$ .
2.  $\Gamma \leftarrow \{T_j \mid T_j \in TS \text{ and } d(T_j) \geq t\}$ .
3. If  $\Gamma = \emptyset$ , then go to Step (9).



4. Let  $T_i$  be the task in  $\Gamma$  such that  $T_i$  has the largest total execution time among all tasks in  $\Gamma$ . (An arbitrary tie-breaking rule can be used to break ties.)  $\Delta \leftarrow \min\{\Theta, o(T_i)\}$ .
5. Construct a task  $T'_i$  with  $d(T'_i) \leftarrow d(T_i)$  and  $e(T'_i) \leftarrow m(T_i) + \Delta$ .
6.  $TS' \leftarrow TS' \cup \{T'_i\}$ .
7.  $\Theta \leftarrow \Theta - \Delta$ .  $t \leftarrow t - (m(T_i) + \Delta)$ .  $TS \leftarrow TS - \{T_i\}$ .
8.  $l \leftarrow l + 1$ . Go to step (2).
9. Use Smith's rule to construct a schedule  $S_b$  for  $TS'$ . □

$K = 10$

$T_i$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$
$d(T_i)$	10	10	16	16	27	27
$m(T_i)$	2	4	6	1	4	3
$o(T_i)$	2	2	1	7	1	4

$T_i$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$
$d(T_i)$	10	10	16	16	27	27
$\alpha(S_b, T_i)$	2	4	6	3	5	7

Figure 4(a). Processing Times Assigned to the Tasks by Algorithm B.

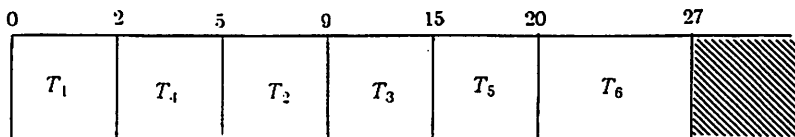


Figure 4(b). The Schedule  $S_b$ .

**Figure 4. Example Task System Illustrating Algorithm B.**

Figure 4 shows an example task system to be scheduled on a single processor. The processing time assigned to each task is shown in Figure 4(a) and the schedule obtained by Algorithm B is shown in Figure 4(b). Using a heap, Algorithm B can be implemented to run in  $O(n \log n)$  time.

For a given task system, let  $S_b$  and  $S_o$  denote the schedules obtained by Algorithm B and an optimal algorithm, respectively. The next theorem shows that there are example task systems such that the performance ratio can approach 2 asymptotically.

**Theorem 3.1.** *There are task systems such that  $MFT(S_b)/MFT(S_o)$  approaches 2 asymptotically.*

**Proof:** Consider the task system  $TS = (\{T_i\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$  with the following  $2m + 1$  tasks, where  $m \geq 2$ ;  $(T_i, Y + (m^2 + 1)X + m(1 - \delta), 1, mX)$  for each  $1 \leq i \leq m$ ,  $(T_i, Y + (m^2 + 1)X + m(1 - \delta), X - \delta, (m - 1)X + \delta)$  for each  $m + 1 \leq i \leq 2m$ , and  $(T_{2m+1}, Y + X + m(1 - \delta), Y, 0)$ , where  $\delta > 2$ ,  $X \geq m\delta$  and  $Y$  is a number much larger than  $X$ . Let  $K = (m^2 - 1)X + m\delta$ .

Let  $S$  be a feasible schedule for  $TS$ . If  $T_{2m+1}$  is scheduled  $i$ th from the last in  $S$ , then  $MFT(S) = iY + C_S$ , where  $C_S$  is a quantity involving  $X$ ,  $m$  and  $\delta$  only. Now, if  $Y$  is a number much larger than  $C_S$ , then  $MFT(S) \approx iY$ . Moreover, in order for  $T_{2m+1}$  to meet its deadline, we must have  $i \geq m + 1$ .

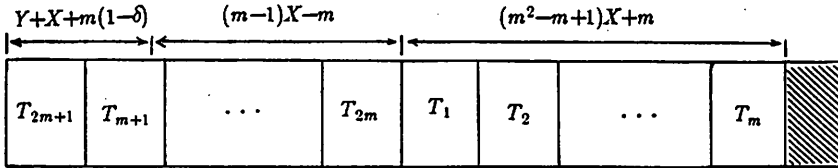


Figure 5(a). The Schedule  $S_b$ .

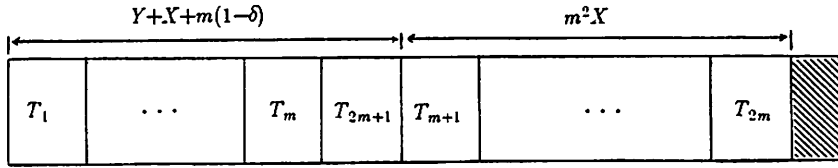


Figure 5(b). The Schedule  $S_o$ .

**Figure 5. Worst-Case Example of Algorithm B.**

Figures 5(a) and 5(b) depict the schedules  $S_b$  and  $S_o$ , respectively. As shown in Figure 5(a), the set  $\{T_1, T_2, \dots, T_m\}$  is scheduled last in  $S_b$ , with  $\alpha(S_b, T_1) = 1 + X$  and  $\alpha(S_b, T_i) = 1 + mX$  for each  $2 \leq i \leq m$ . The

set  $\{T_{m+1}, T_{m+2}, \dots, T_{2m}\}$  is scheduled next, with  $\alpha(S_b, T_i) = X - \delta$  for each  $m+1 \leq i \leq 2m$ . Finally,  $T_{2m+1}$  is scheduled in the first position with  $\alpha(S_b, T_{2m+1}) = Y$ . Since  $T_{2m+1}$  is scheduled  $(2m+1)$ th from the last in  $S_b$ , we have  $\text{MFT}(S_b) \approx (2m+1)Y$ . As shown in Figure 5(b), the set  $\{T_{m+1}, T_{m+2}, \dots, T_{2m}\}$  is scheduled last in  $S_o$ , with  $\alpha(S_o, T_i) = mX$  for each  $m+1 \leq i \leq 2m$ .  $T_{2m+1}$  is scheduled next with  $\alpha(S_o, T_{2m+1}) = Y$ . Finally, the set  $\{T_1, T_2, \dots, T_m\}$  is scheduled with  $\alpha(S_o, T_m) = 1 + X - m\delta$  and  $\alpha(S_o, T_i) = 1$  for each  $1 \leq i \leq m-1$ . Since  $T_{2m+1}$  is scheduled  $(m+1)$ th from the last in  $S_o$ , we have  $\text{MFT}(S_o) \approx (m+1)Y$ . Thus,  $\text{MFT}(S_b)/\text{MFT}(S_o)$  approaches 2 as  $Y$  and  $m$  approach infinity.  $\square$

Before we show that the worst-case performance bound of Algorithm B is 2, we need to introduce the following notations. Let  $T_{x_1}, T_{x_2}, \dots, T_{x_n}$  be the order of the tasks scheduled in  $S_b$  such that  $T_{x_i}$  is the  $i$ th task (from the beginning) scheduled in  $S_b$ . Similarly, let  $T_{y_1}, T_{y_2}, \dots, T_{y_n}$  be the order of the tasks scheduled in  $S_o$ . Let there be  $k$  tasks scheduled in  $S_b$  with their optional parts completely or partially deleted. Clearly, these  $k$  tasks must be the first  $k$  tasks scheduled in  $S_b$ ; i.e., they are  $T_{x_1}, T_{x_2}, \dots, T_{x_k}$ . (Note that it is possible that one of the first  $k-1$  tasks has its optional part partially deleted while the  $k$ th task has its optional part completely deleted. See Figure 4 for example. In this case the task with its optional part partially deleted must have deadline no earlier than that of the  $k$ th task.) Moreover, for each  $k+1 \leq i \leq n$ , we have  $\alpha(S_b, T_{x_i}) = e(T_{x_i})$ . Without loss of generality, we may assume that the tasks are scheduled in  $S_o$  by Smith's rule. Furthermore, by the result in [4], we may assume that  $S_o$  satisfies the properties:  $S_o$  is a nonpreemptive schedule with no idle time,  $\text{ERR}(S_o) = K$ , and there is an index  $l$  such that the first  $l-1$  tasks scheduled in  $S_o$  have no optional part, the last  $n-l$  tasks have full optional parts, and the  $l$ th task has some (possibly all) optional part. In Lemma 3.2, we will show that  $f(S_b, T_{x_i}) < f(S_o, T_{y_i})$  for each  $k \leq i \leq n$ . The next lemma is instrumental in proving this result.

**Lemma 3.1.** *Suppose there is an index  $i$ ,  $k \leq i \leq n$ , such that  $f(S_b, T_{x_i}) < f(S_o, T_{y_i})$ , and let  $u$  be the largest such index. Let  $T_{c_1}, T_{c_2}, \dots, T_{c_z}$  be the  $z$  tasks that are in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$  but not in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$  and let  $T_{d_1}, T_{d_2}, \dots, T_{d_z}$  be the  $z$  tasks that are in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$  but not in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$ . If  $f(S_b, T_{c_i}) \leq f(S_b, T_{c_{i+1}})$  and  $f(S_o, T_{d_i}) \leq f(S_o, T_{d_{i+1}})$  for each  $1 \leq i < z$ , then we have  $\alpha(S_o, T_{d_i}) \leq \alpha(S_b, T_{c_i})$  for each  $1 \leq i < z$ .*

**Proof:** Observe that  $f(S_b, T_{x_i}) \leq f(S_o, T_{y_i})$  for each  $u < i \leq n$  and  $f(S_b, T_{x_u}) > f(S_o, T_{y_u})$ . We prove the lemma by contradiction. Assume there is an index  $i$ ,  $1 \leq i \leq z$ , such that  $\alpha(S_o, T_{d_i}) > \alpha(S_b, T_{c_i})$ , and let  $v$  be the largest such index. Let  $T_{c_v}$  be the  $p$ th task in  $S_b$ , i.e.,  $T_{c_v} = T_{x_p}$  and let  $T_{d_v}$  be the  $q$ th task in  $S_o$ , i.e.,  $T_{d_v} = T_{y_q}$ . Clearly, we have  $u+1 \leq p$ ,

$q \leq n$ . We consider two cases, depending on whether  $p \leq q$  or not.

*Case I:  $p \leq q$ .*

Since  $f(S_o, T_{y_q}) \geq f(S_b, T_{x_q}) \geq f(S_b, T_{x_p})$ , the deadline of  $T_{y_q}$  is no earlier than  $f(S_b, T_{x_p})$ . But  $T_{y_q}$  is not scheduled in  $S_b$  at time  $f(S_b, T_{x_u})$  or after. By the nature of Algorithm B, every task scheduled in  $S_b$  in the time interval  $[f(S_b, T_{x_u}), f(S_b, T_{x_p})]$  must have total execution time at least that of  $T_{y_q}$ . Thus, we have  $\alpha(S_b, T_{x_p}) = e(T_{x_p}) \geq e(T_{y_q}) \geq \alpha(S_o, T_{y_q})$ , and hence,  $\alpha(S_o, T_{d_u}) \leq \alpha(S_b, T_{c_u})$ . This contradicts our assumption that  $\alpha(S_o, T_{d_u}) > \alpha(S_b, T_{c_u})$ .

*Case II:  $p > q$ .*

Since  $f(S_o, T_{y_q}) \geq f(S_b, T_{x_q})$ , the deadline of  $T_{y_q}$  is no earlier than  $f(S_b, T_{x_q})$ . Using the same argument as in Case I, we can show that every task scheduled in  $S_b$  in the time interval  $[f(S_b, T_{x_u}), f(S_b, T_{x_q})]$  must have total execution time at least that of  $T_{y_q}$ . To continue with the proof, we need to define the following notations. Let  $t_1 = f(S_b, T_{x_q})$  and let  $X_1$  be the set of tasks completely executed in the time intervals  $[f(S_b, T_{x_u}), t_1]$  in  $S_b$ . For  $i > 1$ ,  $t_i$  is defined to be the largest deadline of all the tasks in  $X_{i-1}$  and  $X_i$  is defined to be the set of tasks completely executed in the time intervals  $[f(S_b, T_{x_u}), t_i]$  in  $S_b$ . Let  $j > 1$  be the smallest index such that  $X_j = X_{j-1}$ . By the nature of Algorithm B, it is easy to see that every task in  $X_j$  must have total execution time at least that of  $T_{y_q}$ . Let  $T_{x_w} \in X_j$  be the task that finishes last in  $S_b$ , among all tasks in  $X_j$ . Clearly, we have  $f(S_b, T_{x_w}) \leq t_j$ .

We now consider two separate cases, depending on whether  $w \geq p$  or not. If  $w \geq p$ , then we have  $T_{x_p}$  in the set  $X_j$ . Thus, we have  $\alpha(S_b, T_{x_p}) = e(T_{x_p}) \geq e(T_{y_q}) \geq \alpha(S_o, T_{y_q})$ , and hence  $\alpha(S_o, T_{d_u}) \leq \alpha(S_b, T_{c_u})$ . This contradicts our assumption that  $\alpha(S_o, T_{d_u}) > \alpha(S_b, T_{c_u})$ . On the other hand, if  $w < p$ , then we consider the two sets of tasks  $P = \{T_{x_{w+1}}, T_{x_{w+2}}, \dots, T_{x_n}\}$  and  $Q = \{T_{y_{w+1}}, T_{y_{w+2}}, \dots, T_{y_n}\}$ . Among the  $n - w$  tasks in  $P$ , there are  $z - v + 1$  tasks that are in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$  but not in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$ , namely,  $T_{c_u}, T_{c_{u+1}}, \dots, T_{c_z}$ . On the other hand, there are  $z - v$  tasks in  $Q$  that are in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$  but not in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$ , namely,  $T_{d_{u+1}}, T_{d_{u+2}}, \dots, T_{d_z}$ . By the pigeonhole principle, there must be a task  $T' \in Q$  that is not in the set  $P$ . The task  $T'$  must be in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_w}\}$ . Furthermore,  $d(T') \geq f(S_o, T_{y_{w+1}}) \geq f(S_b, T_{x_{w+1}}) > t_j$ . But this contradicts the definitions of  $t_j$  and  $X_j$  as given above. In both cases we obtain a contradiction. Hence the lemma is proved.  $\square$

**Lemma 3.2.** For each  $k \leq i \leq n$ , we have  $f(S_o, T_{y_i}) \geq f(S_b, T_{x_i})$ .

**Proof:** If the lemma were not true, then there would be an index  $i$ ,  $k \leq i \leq n$ , such that  $f(S_o, T_{y_i}) < f(S_b, T_{x_i})$ . Let  $u$  be the largest such index. Let

$T_{c_1}, T_{c_2}, \dots, T_{c_z}$  be the  $z$  tasks that are in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$  but not in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$ , and let  $T_{d_1}, T_{d_2}, \dots, T_{d_z}$  be the  $z$  tasks that are in the set  $\{T_{y_{u+1}}, T_{y_{u+2}}, \dots, T_{y_n}\}$  but not in the set  $\{T_{x_{u+1}}, T_{x_{u+2}}, \dots, T_{x_n}\}$ . By Lemma 3.1, we have  $\alpha(S_o, T_{d_i}) \leq \alpha(S_b, T_{c_i})$  for each  $1 \leq i \leq z$ . Thus, we have  $\sum_{i=u+1}^n \alpha(S_o, T_{y_i}) \leq \sum_{i=u+1}^n \alpha(S_b, T_{x_i})$ . Since  $S_o$  and  $S_b$  have the same schedule length, we have  $f(S_o, T_{y_u}) \geq f(S_b, T_{x_u})$ , contradicting our assumption that it is not.  $\square$

The last lemma implies that the total finishing time of the last  $n - k + 1$  tasks in  $S_b$  is no larger than  $MFT(S_o)$ . If we can show that the total finishing time of the first  $k - 1$  tasks in  $S_b$  is also no larger than  $MFT(S_o)$ , then we immediately have  $MFT(S_b) \leq 2MFT(S_o)$ . The next lemma shows this result.

**Lemma 3.3.**  $\sum_{i=1}^{k-1} f(S_b, T_{x_i}) \leq MFT(S_o)$ .

**Proof:** If  $T_{x_1}, T_{x_2}, \dots$ , and  $T_{x_{k-1}}$  are scheduled in  $S_b$  with their mandatory parts only, then it is clear that  $\sum_{i=1}^{k-1} f(S_b, T_{x_i}) \leq \sum_{i=1}^{k-1} f(S_o, T_{x_i}) \leq MFT(S_o)$ . Thus, we may assume that there is a task  $T_{x_j}$ ,  $1 \leq j \leq k - 1$ , such that  $\alpha(S_b, T_{x_j}) > m(T_{x_j})$ . From the discussions given prior to Lemma 3.1, we know that  $T_{x_k}$  is scheduled in  $S_b$  with its mandatory part only and the deadline of  $T_{x_k}$  is no earlier than  $f(S_b, T_{x_k})$ . Furthermore, we have  $\alpha(S_b, T_{x_j}) \leq \alpha(S_b, T_{x_k}) = m(T_{x_k})$ . Thus, we have  $\sum_{i=1}^{k-1} f(S_b, T_{x_i}) \leq \sum_{i=1}^{j-1} f(S_o, T_{x_i}) + \sum_{i=j+1}^k f(S_o, T_{x_i}) \leq MFT(S_o)$ . The lemma follows immediately.  $\square$

Using Lemmas 3.2 and 3.3, we can prove the main result in this section.

**Theorem 3.2.** For any task system  $TS = (\{T_i\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$  to be scheduled on a single processor, we have  $MFT(S_b) \leq 2MFT(S_o)$ .

**Proof:** We have

$$\begin{aligned} MFT(S_b) &= \sum_{i=1}^{k-1} f(S_b, T_{x_i}) + \sum_{i=k}^n f(S_b, T_{x_i}) \\ &\leq MFT(S_o) + \sum_{i=k}^n f(S_o, T_{y_i}) \text{ (by Lemmas 3.3 and 3.2)} \\ &\leq MFT(S_o) + MFT(S_o) \\ &= 2MFT(S_o). \end{aligned}$$

$\square$

#### 4. Conclusions

In this paper we have given two heuristics for minimizing total flow time for the imprecise computation model. Algorithm A given in Section 2 is

for a set of tasks with a large deadline to be scheduled on  $p \geq 1$  identical processors. It was shown that the worst-case performance bound of Algorithm A is  $3/2$ . Furthermore, we showed that there are example task systems achieving a ratio of  $3/2$  for a single processor and  $5/4$  for multiprocessors. It is still an open question whether  $3/2$  is the best bound for multiprocessors as well.

Algorithm B given in Section 3 is for a set of tasks with different deadlines to be scheduled on a single processor. We showed that the worst-case performance bound of Algorithm B is 2 and the bound is tight. Observe that Algorithm B can produce schedules violating the last condition stated at the beginning of Section 3: there is an index  $l$  such that the first  $l - 1$  tasks scheduled have no optional part, the last  $n - l$  tasks scheduled have full optional parts, and the  $l$ th task has some (possibly all) optional part. See Figure 4 for example. We can always modify Algorithm B so that the final schedule produced satisfies this condition. The modified algorithm performs at least as well as Algorithm B in all cases, and better in some cases. However, the modified algorithm has the same worst-case performance bound as Algorithm B, as the results in Section 3 indicate.

In this paper we have assumed that all tasks have identical ready times. We have also considered the situation where tasks can have arbitrary ready times. Unfortunately, we were not able to come up with a heuristic with a reasonably good worst-case performance bound. For future research, we think it is worthwhile to investigate this issue.

### Acknowledgment

We wish to thank Wei-Kuan Shih for suggesting Algorithm A to us and providing us with some valuable ideas.

### References

- [1] E.K.P. Chong and W. Zhao, Performance Evaluation of Scheduling Algorithms for Dynamic Imprecise Soft Real-Time Computer Systems, *Australian Computer Science Communications* 11 (1989), 329–340.
- [2] J-Y. Chung and J.W.S. Liu, Algorithms for Scheduling Periodic Jobs to Minimize Average Error, *Proc. of the 9th IEEE Real-Time Systems Symposium*, December 1988, 142–151.
- [3] E.G. Coffman, Jr. and P.J. Denning, *Operating Systems Theory*, Prentice Hall, New Jersey, 1973.
- [4] J. Y-T. Leung, T.W. Tam, C.S. Wong and G.H. Young, Minimizing Mean Flow Time with Error Constraint, *Proc. of the 10th IEEE Real-Time Systems Symposium*, December 1989, 1–11.

- [5] K-J. Lin, S. Natarajan and J.W.S. Liu, Concord: A Distributed System Making Use of Imprecise Results, *Proc. of IEEE COMPSAC '87*, October 1987.
- [6] K-J. Lin, S. Natarajan and J.W.S. Liu, Imprecise Results: Utilizing Partial Computations in Real-Time Systems, *Proc. of the 8th IEEE Real-Time Systems Symposium*, December 1987, 210-217.
- [7] J.W.S. Liu, K-J. Lin. and S. Natarajan, Scheduling Real-Time, Periodic Jobs Using Imprecise Results, *Proc. of the 8th IEEE Real-Time Systems Symposium*, December 1987, 252-260.
- [8] R. McNaughton, Scheduling with Deadlines and Loss Functions, *Management Science* 6 (1959), 1-12.
- [9] W-K. Shih, J.W.S. Liu and J-Y. Chung, Algorithms for Scheduling Imprecise Computations with Timing Constraints, *SIAM J. Computing* 20 (1991), 537-552.
- [10] W-K. Shih, J.W.S. Liu, J-Y. Chung and D.W. Gillies, Scheduling Tasks with Ready Times and Deadlines to Minimize Average Error, *ACM Operating Systems Review*, July 1989.
- [11] W.E. Smith, Various Optimizers for Single-Stage Production, *Naval Research Logistics Quarterly* 3 (1956), 59-66.