

Short Paper

Algorithms for Imprecise Tasks With 0/1-Constraints^{*}

KUN-MING YU

*Department of Computer Science and Information Engineering
Chung-Hua University
Hsinchu, Taiwan 300, R.O.C.*

We consider the problem of preemptively scheduling a set of imprecise computation tasks on a single processor, with the 0/1-constraint. In the imprecise computation model, each task consists of two parts, mandatory and optional, with the mandatory part required to be completed while the optional part can be left uncompleted. If a task has an optional part that is unfinished, then it incurs an *error* equal to the processing time of its unfinished portion. In the 0/1 constraint environment, each optional subtask is either fully scheduled or entirely discarded. Two performance metrics are considered: (1) the number of *imprecisely scheduled* tasks; (2) the total *error*. For the problem of minimizing the number of imprecisely scheduled tasks, it has been shown that it can be solved in $O(n^3)$ -time. Since the time complexity is relatively high. We propose two $O(n^2)$ -time algorithms for two special cases. On the other hand, it is known that the problem of minimizing the total error is NP-hard. We present an $O(n^2)$ time algorithm to solve the problem when tasks have optional parts of the same length.

Keywords: real-time system, imprecise computation task, preemptive scheduling, mandatory subtask, optional subtask, total error, imprecisely scheduled tasks, NP-hard

1. INTRODUCTION

Meeting deadline constraints is an essential consideration for *real-time* tasks. In a *real-time* system, every task is associated with a *release time*, *processing time* and *deadline*. A task's execution cannot begin before its *release time*, and the task needs to receive a processor time equal to its *processing time* before its *deadline*. It is vital for every *real-time* task to meet its timing constraint; that is, such a task should be scheduled to be executed after its release time and to be finished before its deadline. Such a schedule is called a *feasible schedule*. If a system cannot produce a feasible schedule for a given set of tasks, then the results produced by the system are of little or no value. Unfortunately, for many real-time applications, it is not always possible to schedule all the tasks so as to meet their timing constraints, a situation that occurs quite often when a system is heavily loaded. To overcome this problem, certain less important tasks can be given up in order

Received August 29, 1998; revised September 30 & December 2, 1999; accepted January 18, 2000.

Communicated by Shing-Tsaan Huang.

^{*}Research supported in part by the NSC Grant NSC 82-0408-E-216-003.

to meet the deadlines for more important ones. Another approach is to reduce the amount of computation time required to produce these tasks, and such a trade off can be realized by using *imprecise computation* techniques.

The *Imprecise Computation Model* was initially introduced by Lin *et al.* [12]. Subsequently, scheduling algorithms were added to minimize the total error for periodic tasks [13, 14]. In this model, a task is logically decomposed into two subtasks, a *mandatory* subtask and an *optional* subtask. Every mandatory subtask must be completed before its deadline while each optional subtask can be left unfinished. If an optional subtask is not completed by its deadline, an error is said to occur, which is equal to the processing time of its unfinished portion.

In the Imprecise Computation Model, a task T_i is represented by the quadruple $(r(T_i), d(T_i), m(T_i), o(T_i))$, where $r(T_i)$, $d(T_i)$, $m(T_i)$ and $o(T_i)$ denote its *release time*, *deadline*, *mandatory subtask processing time*, and *optional subtask processing time*, respectively. For simplicity, let $M_i(O_i)$ denote the mandatory (optional) subtask of T_i . We use $e(T_i)$ to denote the total processing time of T_i ; i.e., $e(T_i) = m(T_i) + o(T_i)$. If the importance of each task is not the same, a weight can be associated with each task; such a task system is called a *weighted* task system. A schedule is said to be *preemptive* if the tasks are allowed to be preempted; otherwise, it is *nonpreemptive*. A schedule is *M-feasible* for a given task system if each mandatory subtask in the task system is completed by its deadline; a task system is *M-feasible* if there is an M-feasible schedule for it. The feasibility test for a task system on a single processor can be determined in $O(n \log n)$ [14], and in a maximum of $O(n^2 \log^2 n)$ time for a multi-processor system [15]. In this paper, we will assume that all task systems are M-feasible, and we will only focus on preemptive scheduling.

For the problem of minimizing the total error, Liu *et al.* [13] first proposed an $O(n^2 \log^2 n)$ -time algorithm for a multi-processor system. Recently, Shih, Liu and Chung [14] and Leung, Yu and Wei [10] have given an $O(n \log n)$ - time algorithm for a single processor. Considering the weighted case for the multi-processor system, Shih *et al.* [15] showed that the problem can be reduced to a minimum-cost-maximum-flow problem and can be solved in $O(n^2 \log^3 n)$ -time. For a single processor, Shih *et al.* [14] presented a faster algorithm that runs in $O(n^2 \log n)$ time. Recently, Leung, Yu and Wei [10] have proposed an $O(n \log n + kn)$ -time algorithm, where k is the number of distinct weights in the task system.

For application purposes, Shih *et al.* [14] added a constraint to the Imprecise Computation Model, in which each optional subtask is either fully executed or entirely discarded. We refer to this kind of constraint as the 0/1-constraint. Along with the 0/1-constraint, two problems were proposed in that paper [14]. The first one was to minimize the total error, and the second was to minimize the number of *imprecisely scheduled* tasks (i.e., tasks whose optional subtasks have been discarded completely). For a single processor, the problem of minimizing the total error is NP-complete because it can be easily reduced from a partition problem, even when all the tasks have the same release time and deadline. Also, Shih *et al.* [14] presented an $O(n^2 \log n)$ -time algorithm to solve the problem of minimizing the number of imprecisely scheduled tasks when the optional subtasks have identical processing time. We will give an $O(n^2)$ algorithm for the same problem in this article.

For the problem of minimizing the number of imprecisely scheduled tasks, Shih *et al.* [14] showed that when the optional subtasks have identical processing time, this problem can be solved in polynomial time. Recently, Ho *et al.* [6] applied Lawler's algorithm [9] to this problem and gave an $O(n^5)$ -time optimal algorithm for the general case of this problem. In this article, we study two special cases for the problem of minimizing the number of imprecisely scheduled tasks, i.e., where all the tasks have release times and deadlines that have either similar or opposite order. We give $O(n^2)$ -time complexity for both cases. Also, we propose an $O(n^2)$ -time optimal algorithm for this problem when the optional subtasks have the same processing time.

The following portions of this paper are organized as follows. In the next section, we discuss the basic workload model and study the problem of minimizing the number of imprecisely scheduled tasks. Also, in that section, we review the algorithm given by Hochbaum and Shamir [7]. In Section 3, we present two algorithms for special cases of the problem of minimizing the number of imprecisely scheduled tasks. In Section 4, the problem of minimizing the total numbers of errors is examined. We give an $O(n^2)$ -time algorithm for this problem when the optional subtasks have the same processing time. Also, we show that this $O(n^2)$ -time algorithm can be applied to an imprecisely scheduled problem with the same constraints. Finally, we draw some conclusions in the last section.

2. IMPRECISELY SCHEDULED TASKS

In this section, we give optimal algorithms for some special cases of scheduling a set of imprecise computation tasks with the 0/1-constraint on a single processor. In Section 3, we discuss cases where the input task system has the following properties: (1) release times and deadlines are similarly ordered for all tasks and (2) release times and deadlines have opposite order for all tasks. In both cases, we need to apply the algorithm of Hochbaum and Shamir [7] to obtain an optimal schedule after deciding on the precisely scheduled tasks in the optimal schedule. Therefore, we now describe the algorithm of Hochbaum and Shamir, which solves the unweighted case in $O(n \log n)$ time. We denote their algorithm as Algorithm HS for brevity.

Algorithm HS assumes that the tasks have been indexed in nonincreasing order of release times, i.e., $r(T_1) \geq r(T_2) \geq \dots \geq r(T_n)$. Let $0 = u_0 < u_1 < \dots < u_p = \max_{1 \leq i \leq n} (d(T_i))$ be the $p+1$ distinct integers obtained from the multiset $\{r(T_1), r(T_2), \dots, r(T_n), d(T_1), d(T_2), \dots, d(T_n)\}$. These $p+1$ integers divide the time frame into p segments: $[u_0, u_1], [u_1, u_2], \dots, [u_{p-1}, u_p]$. Algorithm HS schedules tasks in nonincreasing order of release times. When a task is scheduled, it is assigned from the latest segment $[u_{a-1}, u_a]$, in which it can be nontardy, until the earliest segment $[u_b, u_{b+1}]$, in which u_b is equal to its release time, with the maximum number of time units assigned in each segment. The output of Algorithm HS is an $n \times p$ matrix, S , where S_{ij} is the number of time units of task T_i scheduled in segment $j([u_{i-1}, u_i])$. The output matrix S is of size $O(n^2)$. However, it was showed in [7] that the number of nonzero entries in S is $O(n)$. Below is a formal description of the algorithm.

Algorithm HS

(1) **For** $i = 1, \dots, p$ **do** $l_i \leftarrow u_i - u_{i-1}$.

```

(2) For  $i = 1, \dots, n$  do
    Find  $a$  satisfying  $u_a = d(T_i)$  and  $b$  satisfying  $u_b = r(T_i)$ .
    For  $j = a, a-1, \dots, b+1$  do
         $\delta \leftarrow \min \{l_j, e(T_i)\}$ .
         $S_{ii} \leftarrow \delta, l_i \leftarrow l_j - \delta, e(T_i) \leftarrow e(T_i) - \delta$ .
    repeat

```

□

By using the UNION-FIND technique of Gabow and Tarjan [3], we find that the complexity of Algorithm HS is $O(n \log n)$. The reader can refer to [7] for details. We say a schedule is an HS-schedule if it is produced by Algorithm HS.

3. TWO OPTIMAL ALGORITHMS FOR IMPRECISELY SCHEDULED TASKS

3.1 Release Times and Deadlines With Opposite Order

We consider here the special case of the imprecisely scheduled task problem in which release times and deadlines have opposite order; i.e., the tasks can be numbered so that

$$\begin{aligned} r(T_1) &\geq r(T_2) \geq \dots \geq r(T_n), \\ d(T_1) &\leq d(T_2) \leq \dots \leq d(T_n). \end{aligned}$$

In this problem, all the release times, deadlines and processing times may be ratio numbers. This problem can be formulated as a variation of the knapsack problem as follows.

We have $2n$ new tasks, T'_i , $1 \leq i \leq 2n$, in the task system TS' . (For each original task T_i , $1 \leq i \leq n$, we create two tasks – an M -task, T'_{2i-1} , with processing time $e(T'_{2i-1}) = m(T_i)$; and an O -task, T'_{2i} , with processing time $e(T'_{2i}) = o(T_i)$.) Every M -task has weight $n+1$, and every O -task has weight 1. Our goal is to find a feasible set of tasks with the maximum weight. Let $x_i = 1$ if task i is chosen to be in the feasible set; otherwise, $x_i = 0$. Then the above problem can be expressed as follows:

$$\begin{aligned} &\text{maximize } \sum w_i x_i \\ &\text{subject to} \\ &\quad e(T'_1) x_1 \leq d(T'_1) - r(T'_1) \\ &\quad e(T'_1) x_1 + e(T'_2) x_2 \leq d(T'_2) - r(T'_2) \\ &\quad e(T'_1) x_1 + e(T'_2) x_2 + e(T'_3) x_3 \leq d(T'_3) - r(T'_3) \\ &\quad \quad \quad \cdot \\ &\quad \quad \quad \cdot \\ &\quad e(T'_1) x_1 + e(T'_2) x_2 + e(T'_3) x_3 + \dots + e(T'_{2n}) x_{2n} \leq d(T'_{2n}) - r(T'_{2n}) \\ &\text{and } x_i \in \{0,1\}, 1 \leq i \leq 2n. \end{aligned}$$

We can then use the following dynamic programming method to solve this problem.

Let TBL1 be a $2n \times (W+1)$ table, where $W = \sum_{i=1}^{2n} w(T'_i)$. Each entry of TBL1, say

$TBL1(i, j)$, contains the minimum processing time for the tasks in $\{T'_1, T'_2, T'_3, \dots, T'_i\}$ such that these tasks are feasibly scheduled in the interval $[r_i, d_i]$ and the weight is equal to j . (Initially, we fill in all the entries of TBL1 with an infinite value, ∞ .) The maximum weight of the M-feasible is equal to the value of j of the entry $TBL1(2n, j)$, where $TBL1(2n, j)$ is the rightmost non-infinite value entry of the last row in TBL1. The following algorithm shows how TBL1, as well as an optimal schedule, can be constructed.

Algorithm A

Input: A task system $TS = (\{T_i\}, \{r(T_i)\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$, where $1 \leq i \leq n$.

Moreover, the tasks can be numbered so that $r(T_1) \geq r(T_2) \geq \dots \geq r(T_n)$ and $d(T_1) \leq d(T_2) \leq \dots \leq d(T_n)$

Output: An optimal schedule with a minimum number of imprecisely scheduled tasks.

Method:

- (1) Transform the input task system TS into a new task system $TS' = (\{T'_i\}, \{r(T'_i)\}, \{d(T'_i)\}, \{e(T'_i)\}, \{w(T'_i)\})$, where $1 \leq i \leq 2n$. In TS' , $e(T'_{2i-1}) = m(T_i)$, $e(T'_{2i}) = o(T_i)$, where $1 \leq i \leq n$. Also, there is a weight associated with every task in TS' as follows: $w(T'_i) = n+1$ when i is odd; otherwise $w(T'_i) = 1$. Furthermore, $r(T'_i) \geq r(T'_{i+1})$ and $d(T'_i) \leq d(T'_{i+1})$ for all i , $1 \leq i \leq 2n-1$.
- (2) $TBL1(i, 0) \leftarrow 0$ for all $1 \leq i \leq 2n$.
 $TBL1(i, j) \leftarrow \infty$ for all $1 \leq i \leq 2n$, $1 \leq j \leq W$.
- (3) $TBL1(1, w(T'_1)) = e(T'_1)$.
- (4) For i from 2 to $2n$ do
 For j from $\lfloor i/2 \rfloor * (n+1)$ to $\lfloor i/2 \rfloor * (n+1) + \lfloor i/2 \rfloor$ do
 $TBL1(i, j) = \min\{TBL1(i-1, j), TBL1(i-1, j-w(T'_i)) + e(T'_i)\}$
 if $TBL1(i-1, j-w(T'_i)) + e(T'_i) \leq d(T'_i) - r(T'_i)$
 $= TBL1(i-1, j)$ otherwise.
- (5) Maximum j , such that $TBL1(2n, j) \neq \infty$, is the maximum weight of the feasible set for the input task system TS' .
- (6) Use the pointer stored in the table TBL1 to find all the tasks selected in the optimal task set S_{opt} .
- (7) Run Algorithm HS using S_{opt} as the input to produce an optimal schedule. \square

We will now show that Algorithm A correctly constructs the table TBL1 and produces an optimal schedule. In Step (1), Algorithm A transforms the input task systems TS into a new task system TS' according to the method described above. In Step (4), the algorithm computes the value of the entries from rows 2 to $2n$ in the table. The algorithm computes row i from row $i-1$ for $2 \leq i \leq 2n$ as follows. For each entry $TBL1(i, j)$, $\lfloor i/2 \rfloor * (n+1) \leq j \leq \lfloor i/2 \rfloor * (n+1) + \lfloor i/2 \rfloor$, the algorithm compares the value of $TBL1(i-1, j-w(T'_i)) + e(T'_i)$ and $TBL1(i-1, j)$, selects the lower value and assigns to the entry $TBL1(i, j)$. Since TS is an M-feasible task system, the T'_{2i-1} tasks should be included in the final schedule. Step (5) simply finds the maximum weights of the input task system. By storing pointers in the table, Algorithm A can use the pointer stored in TBL1 to de-

cide the optimal task set S_{opt} in Step (6). Then, we apply Algorithm HS to S_{opt} , and an optimal schedule is achieved in Step (7).

Each entry in the table TBL1 is computed in constant time, and the total number of computed entries in TBL1 is $2n \lceil i/2 \rceil$, where $1 \leq i \leq 2n$. Thus, the time complexity of Algorithm A is $O(n^2)$ time. From the above discussions, we have the following theorem.

Theorem 3.1: Algorithm A gives the minimum weighted number of late tasks, and its time complexity is $O(n^2)$.

As described above, each M -task in the task system TS has weight $n + 1$, and every O -task in the task system TS' has weight 1. It is impossible for Algorithm A to give up any M -task in favor of scheduling some O -tasks. Therefore, Algorithm A will schedule the all mandatory subtasks in the task system TS and produce a minimum number of imprecisely scheduled tasks. Furthermore, Algorithm A runs in $O(n^2)$ time. Thus, we have Theorem 3.2.

Theorem 3.2: Algorithm A gives the minimum number of imprecisely scheduled tasks, in which all the tasks' release time and deadlines have opposite order, and its time complexity is $O(n^2)$.

3.2 Similarly Ordered Release Time and Deadline

In the second case, where the tasks have similarly ordered release times and deadlines, i.e.,

$$\begin{aligned} r(T_1) &\leq r(T_2) \leq \dots \leq r(T_n), \\ d(T_1) &\leq d(T_2) \leq \dots \leq d(T_n), \end{aligned}$$

all the release times, deadlines and processing times may be ratio numbers.

For each task T_i , $1 \leq i \leq n$, we create two new tasks, an M -task T'_{2i-1} with processing time $e(T'_{2i-1}) = m(T_i)$ and an O -task T'_{2i} with processing time $e(T'_{2i}) = o(T_i)$. Thus, we have $2n$ new tasks, T'_i , $1 \leq i \leq 2n$, in the new task system TS' . Every M -task has weight $n + 1$, and every O -task has weight 1. Our goal is to find a feasible set of tasks with maximum weight. Similar to the previous case, we can use the dynamic programming method to solve this problem. Let TBL2 be a $2n \times (W+1)$ table, where $\text{TBL2}(i, j)$ contains the minimum processing time after time $r(T'_i)$ with respect to the set of all feasible tasks, $S \subseteq \{T'_1, T'_2, T'_3, \dots, T'_i\}$, and where the feasible set has weight j . Let $\text{TBL2}'(i, j)$ denote the minimum processing time after time $r(T'_{i+1})$ with respect to the set of all feasible tasks, $S \subseteq \{T'_1, T'_2, T'_3, \dots, T'_i\}$, i.e., $\text{TBL2}'(i, j) = \max \{0, \text{TBL2}(i, j) - (r(T'_{i+1}) - r(T'_i))\}$. Therefore, we can calculate the value of $\text{TBL2}(i+1, j)$ by choosing the smaller value of $\text{TBL2}'(i, j)$ and $\text{TBL2}'(i, j - w(T'_{i+1})) + e(T'_{i+1})$. The maximum weight of the task system TS' is found by means of the largest w such that $\text{TBL2}(i, w)$ is finite, where $0 \leq w \leq W$. (Initially, we assign a value of ∞ to all the entries in TBL2, except for the first column, i.e., $\text{TBL2}(i, 0)$.) Table TBL2 is constructed as the following algorithm.

Algorithm B

Input: A task system $TS = (\{T_i\}, \{r(T_i)\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$, where $1 \leq i \leq n$.

Moreover, the tasks can be numbered so that $r(T_1) \leq r(T_2) \leq \dots \leq r(T_n)$ and $d(T_1) \leq d(T_2) \leq \dots \leq d(T_n)$.

Output: An optimal schedule with the minimum number of imprecisely scheduled tasks.

Method:

- (1) Transform the input task system TS into a new task system $TS' = (\{T'_i\}, \{r(T'_i)\}, \{d(T'_i)\}, \{e(T'_i)\}, \{w(T'_i)\})$, where $1 \leq i \leq 2n$.
- (2) $TBL2(i, 0) \leftarrow 0$ for all $1 \leq i \leq 2n$.
 $TBL2(i, j) \leftarrow \infty$ for all $1 \leq i \leq 2n, 1 \leq j \leq W+1$.
- (3) $TBL2(1, w(T'_1)) \leftarrow e(T'_1)$, and
 $TBL2'(1, w(T'_1)) \leftarrow \max\{0, TBL2(1, w(T'_1)) - (r(T'_2) - r(T'_1))\}$
- (4) For i from 2 to $2n$ do
 For j from $\lfloor i/2 \rfloor * (n+1)$ to $\lfloor i/2 \rfloor * (n+1) + \lfloor i/2 \rfloor$ do
 $TBL2(i, j) = \min\{TBL2'(i-1, j), TBL2'(i-1, j - w(T'_i)) + e(T'_i)\}$
 if $TBL2'(i-1, j - w(T'_i)) + e(T'_i) \leq d(T'_i) - r(T'_i)$
 $= TBL2'(i-1, j)$ otherwise.
 $TBL2'(i, j) = \max\{0, TBL2(i, j) - (r(T'_{i+1}) - r(T'_i))\}$
- (5) Maximize j , such that $TBL2(2n, j) \neq \infty$ is the maximum weight of the feasible set for the input task system TS' .
- (6) Use the pointer stored in Table $TBL2$ to find the optimal task set S_{opt} .
- (7) Run Algorithm HS using S_{opt} as the input to produce an optimal schedule. \square

In Algorithm B, the role of Step (1) is to transform the input task system TS into a new task system TS' according to the rule we have previously described. Similar to Algorithm A, when an entry in $TBL2$, eg., $TBL2(i, j)$, has a finite value, then there is a schedule for the first i tasks such that they are scheduled in the time interval $[r(T'_i), d(T'_i)]$ and have processing time $TBL2(i, j)$, after $r(T'_i)$, and weight j . In Step (4), the algorithm calculates the value of $TBL2'(i, j)$ for every value of i in $TBL2$, proceeding from row 2 to row $2n$, by choosing the minimal values of $TBL2'(i-1, j)$ and $TBL2'(i-1, j - w(T'_i)) + e(T'_i)$ or by setting the value of the entry to $TBL2'(i-1, j)$. In the next step, the maximum weighted number of tasks or the minimum weighted number of late tasks is determined for the task system TS' . In Step (6), the optimal task set S_{opt} is generated by using the pointer stored in $TBL2$. In the last step, Algorithm B produces an optimal schedule for the input task system TS by applying Algorithm HS to S_{opt} .

We then evaluate the time complexity for Algorithm B. Computing the value of the entry can be done in constant time, and the total number of computed entries in $TBL2$ is $2n * \lfloor i/2 \rfloor$, where $1 \leq i \leq 2n$. Therefore, the time complexity of Algorithm B is $O(n^2)$ -time. Similar to Algorithm A, we can construct an optimal schedule by storing pointers in the table. We conclude our discussion with Theorem 3.3.

Theorem 3.3: Algorithm B gives the minimum weighted number of late tasks, and its time complexity is $O(n^2)$.

Since every M -task in TS' has weight $n+1$ and every O -task in TS' has weight 1, it is not possible for Algorithm B to drop any M -task in order to support scheduling of some

O -tasks(at most n). Thus, Algorithm B will schedule all mandatory subtasks in the original task system TS and produce a minimum of imprecisely scheduled tasks. Also, Algorithm B has $O(n^2)$ time complexity. Therefore, we have Theorem 3.4 as follows.

Theorem 3.4: Algorithm B gives the minimum number of imprecisely scheduled tasks, in which all the tasks' release times and deadlines are similarly ordered, and their time complexity is $O(n^2)$.

4. MINIMIZING THE TOTAL NUMBER OF ERRORS

The problem of minimizing the total number of errors in the imprecise computation model can be solved in $O(n \log n)$ time [10, 13]. After adding the 0/1 constraints to this problem, it becomes an NP-complete problem. The proof of its NP-completeness is relatively easy since it can be reduced from the Partition problem [4]. Since the problem of minimizing the total number of errors is difficult, we will focus on a special case in which all the optional subtasks have the same processing time. Previously, Shih [13] presented a similar problem using an $O(n \log n)$ -time and $O(n^2 \log n)$ -time algorithm for the special case in which all the tasks have the same release time and arbitrary release time, respectively. In the following, we give an $O(n^2)$ -time algorithm for the later case.

Let TS be a task system with n tasks. Let S be an M-feasible schedule for the task system TS and let \mathbf{M} denote the set of all mandatory subtasks in TS. We use $E(T_{i,S})$ to denote the amount of processor time assigned to task T, $1 \leq i \leq n$, in schedule S. The symbols $\epsilon(T_{i,S})$ denote the error of T_i in S, i.e., $\epsilon(T_{i,S}) = e(T_i) - E(T_{i,S})$. The total error of S, denoted by $\epsilon(S)$, is defined as $\sum_{i=1}^n \epsilon(T_{i,S})$. The minimum total error of TS denoted, by $\epsilon(TS)$, is defined as $\min \{\epsilon(S), \text{ where } S \text{ is an M-feasible schedule for TS}\}$. Let PS denote the set of precisely scheduled tasks in TS during execution of the proposed algorithm. This assumes that the tasks have been indexed in nondecreasing order of release time, with tasks having the same release time in nonincreasing order of deadlines. Our algorithm, referred to as Algorithm C, uses Algorithm HS as a subroutine, and it works as follows. First, PS is initialized to be the set including all mandatory subtasks since TS is an M-feasible system based on our assumption. The following process is iterated n times. In the i^{th} iteration, Algorithm HS is applied to compute the minimum total error for the task system TS' obtained from TS by setting the processing times of the optional subtasks of all tasks not in $PS \cup \{T_i\}$ to be zero. If $\epsilon(TS') = 0$, then the algorithm includes T_i 's optional subtask in the task set PS; otherwise, it does not. A formal description of Algorithm C is given below.

Algorithm C

Input: A task system $TS = (\{T_i\}, \{r(T_i)\}, \{d(T_i)\}, \{m(T_i)\}, \{o(T_i)\})$, where $o(T_i) = \delta$ for all $i, 1 \leq i \leq n$. The tasks are assumed to be in nondecreasing order of release times, with tasks having the same release time in nonincreasing order of deadlines.

Output: An optimal schedule S_{opt} with the minimum number of total errors satisfying 0/1-constraints.

Method:

- (1) $PS \leftarrow M$.
- (2) For $i = 1, 2, \dots, n$ do
 Use Algorithm HS to construct a schedule \hat{S} for the task system TS' obtained from TS by setting the processing time of the optional subtasks of all tasks not in $PS \cup \{Ti\}$ to be zero. If $\epsilon(TS') = 0$, then $PS \leftarrow PS \cup O_i$.
- (3) Use Algorithm HS to construct an optimal schedule S_{opt} for the task set PS . \square

Clearly, S_{opt} satisfies the 0/1 constraint. We then examine the time complexity of Algorithm C. Observe that Algorithm C utilizes Algorithm HS to construct schedules for various subsets of TS . With initial sorting of the release times and deadlines of all tasks in TS , we can determine in linear time whether or not $\epsilon(TS') = 0$. Moreover, Algorithm HS only needs linear time to construct a schedule after the order is obtained.

Step (1) of Algorithm C takes linear time, and Step (2) is iterated n times. Each iteration of Step (2) takes only linear time after initial sorting of the tasks in nonincreasing order of the release times. Therefore, the time complexity of Algorithm C is $O(n^2)$.

From the above discussions, we have the following theorem.

Theorem 4.1: Algorithm C has a worst-case time complexity of $O(n^2)$.

The correctness proof of Algorithm C can be found in [14], to which the reader can refer for details. When all optional subtasks have equal processing time in the input task system, the problem of minimizing the number of imprecisely scheduled tasks is identical to the problem of minimizing the total number of errors with 0/1 constraint. Therefore, we have the following theorem:

Theorem 4.2: Algorithm C always produces an M -feasible schedule with the minimum number of imprecisely scheduled tasks.

5. CONCLUSIONS

In this paper, we have studied the problem of minimizing the number of imprecisely scheduled tasks. We have presented two $O(n^2)$ -time algorithms, Algorithm A and Algorithm B, for two special cases. Algorithm A is optimal for the task system in which input tasks have release times and deadlines in opposite order. Algorithm B generates an optimal schedule for the task system in which the input tasks have similarly ordered release times and deadlines.

We have also considered the problem of minimizing the total number of errors with the 0/1 constraint, and presented an $O(n^2)$ -time algorithm, Algorithm C, to solve the case where optional subtasks have the same processing time in the input task system. We have also shown that the problem of minimizing the number of imprecisely scheduled tasks when all optional subtasks have equal processing time can be solved using Algorithm C.

ACKNOWLEDGEMENT

We would like to thank the anonymous referees, whose valuable comments helped to improve the content of this paper.

REFERENCES

1. J. Blazewicz, "Minimizing mean weighted information loss with preemptible tasks and parallel processors," *Tech. Sci. Inform.*, Vol. 3, 1984, pp. 415-420.
2. J. Blazewicz and G. Finke, "Minimizing mean weighted execution time loss on identical and uniform processors," *Information Processing Letters*, Vol. 24, 1987, pp. 259-263.
3. H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," *Journal of Comput. System Sci.*, Vol. 30, 1985, pp. 209-221.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
5. R. L. Graham, E. L. Lawler, J. K. Lenstra, and Rinnooy Kan, A. H. G., "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Ann. Discrete Math.*, Vol. 5, 1979, pp. 287-326.
6. K. I-J Ho, Joseph Y-T. Leung, and W-D. Wei, "Scheduling imprecise computation tasks with 0/1-constraint," *Discrete Applied Mathematics*, Vol. 78, 1997, pp. 117-132.
7. D. S. Hochbaum and R. Shamir, "Minimizing the number of tardy job unit under release time constraints," *Discrete Applied Mathematics*, Vol. 28, 1990, pp. 45-57.
8. W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, Vol. 21, 1974, pp. 177-185.
9. E. L. Lawler, "New and improved algorithms for scheduling a single machine to minimize the weighted number of late jobs," Preprint, Computer Science Division, University of California, 1989.
10. J. Y-T. Leung, V. K-M. Yu, and W-D. Wei, "Minimizing the weighted number of tardy task units," *Discrete Applied Mathematics*, Vol. 51, 1994, pp. 307-316.
11. K-J. Lin, S. Natarajan, and J. W. S. Liu, "Concord: a distributed system making use of imprecise results," in *Proceedings of COMPSAC '87*, 1987.
12. K-J. Lin, S. Natarajan, and J. W. S. Liu, "Imprecise results: utilizing partial computation in real-time systems," in *Proceedings of IEEE 8th Real-Time Systems Symposium*, 1987, pp. 210-217.
13. J. W. S. Liu, W-K Shih, A. C-S. Yu, J-Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," *IEEE Computer*, Vol. 24, No. 5, 1991, pp. 58-68.
14. W-K. Shih, J. W. S. Liu, and J-Y. Chung, "Algorithms for scheduling imprecise computations with timing constraints," *SIAM Journal of Computing*, Vol. 20, No. 3, 1991, pp. 537-552.
15. 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*, 1989, pp. 14-28.

Kun-Ming Yu (游坤明) received the B.S. degree in Chemical Engineering from National Taiwan University in 1981, and the M.S. and Ph.D. degrees in Computer Science from the University of Texas at Dallas in 1988 and 1991, respectively. From August 1993 to July 1996, he was the head of and an associate professor in the Department of Information Management, Chung-Hua Polytechnic Institute. From 1995 to 1996, he was the Director of the Computer Center at Chung-Hua Polytechnic Institute. He is currently an associate professor and chair of the Department of Computer Science and Information Engineering, Chung-Hua University. His research interests include load balancing, internet-based applications, distributing and parallel computing, and computer algorithms.