Journal of the Operations Research
Society of Japan
Vol. 31, No. 1, March 1988

APPROXIMATE AND EXACT ALGORITHMS FOR SCHEDULING INDEPENDENT TASKS ON UNRELATED PROCESSORS

Kazumiti Numata
The University of Electro-Communications

(Received March 12, 1987; Revised July 27, 1987)

Abstract The problem to schedule n independent tasks nonpreemptively on m unrelated processors with the objective of minimizing the finishing time is considered. For the case of m=2, an approximate algorithm which has a worst-case performance ratio $1+\epsilon$ and runs in time $O(n\log n)$ is proposed. For general m, by restricting the number of task types to k, a polynomial time (in n, m) exact algorithm is presented.

1. Introduction

We are given a set $\mathcal{T}=\{T_1,T_2,\ldots,T_n\}$ of $n \geqq 2$ independent tasks, a set $\mathcal{P}=\{P_1,P_2,\ldots,P_m\}$ of $m \geqq 2$ unrelated processors, and a function $\mu(\cdot)$ mapping from $\mathcal{T}\times\mathcal{P}$ into the set of positive integers. There is no precedence relations among tasks. A processor can work on only one task at a time, and a task can be worked on by any (one) processor. Tasks are processed nonpreemptively, i.e., once a task having started execution it will not be interrupted until its completion. The value $\mu(T_i,P_j)$, written shortly μ_{ij}, denotes the execution time of task T_i on processor P_j, where we assume without loss of generality that the times are given in integral multiples of a unit time. The "unrelated" processors system which we treat is a generalization of the identical processors system where μ_{ij} is constant with respect to j i.e. is equal to r_i (execution requirement of task T_i) and of the uniform processors system where by using the notion of processor's speed(s_j) μ_{ij} is expressed as r_i/s_j. In the unrelated case, processor's speed may vary according to the task being executed. Rather, it models the heterogeneous multiprocessor system where respectively specialized processors can execute the each type of tasks more efficiently than others.

The problem is to schedule $\mathcal{T}$ on $\mathcal{P}$ so that the total time taken to process $\mathcal{T}$ is minimized. Such a schedule is called an optimum schedule. The decision problem of determining whether $\mathcal{T}$ can be processed within a given finishing time is known to be *NP*-complete even in the case of two(m=2) identical processors [1, 4]. Hence it is unlikely that the polynomial time exact algorithm can be found to solve the problem. So the investigation has been directed to the fast approximate algorithms and the error analyses, or realistic simplifications of the problem.

For the identical and uniform processors system, simple heuristic algorithms such as LPT or MULTIFIT have been intensively studied [5, 2], but these seem not to be directly applicable to the unrelated processors system. For this system, polynomial time approximate algorithms were first studied by Ibarra and Kim [7]. They presented an $n\log n$ time δ-approximate (with fixed worst-case performance ratio $1+\delta$) algorithm for m=2 processors case, which is guaranteed to be at most $\frac{1+\sqrt{5}}{2}$ times worse than the optimum ($\delta \cong 0.6$). Horowitz and Sahni [6] proposed an ε-approximate algorithm of time complexity $O(n^{2m}/\varepsilon)$ which can generate schdules arbitrarily close to the optimum for general m. However the running time of their algorithm rapidly grows larger as desired relative error bound, ε, gets smaller. Davis and Jaffe [3] presented polynomial time approximate algorithms for the general m, and proved them to be at most $2\sqrt{m} \sim 1.5\sqrt{m}$ times worse than the optimum. Recently, Potts [10] developed a linear programming based heuristic for general m. This algorithm has a worst-case performance ratio of 2 for $m>3$, and a modified version of it for m=2 has a ratio 1.5. As for the problem simplification, Leung [8] considered the identical processors system under the restriction that the number of different execution times is restricted to $k(k \ll n)$, and presented a polynomial time exact algorithm.

In this paper we first propose the new approximate algorithm for unrelated processors system which runs faster than [6] and generates more accurate schedules than [7] or [10]. Next, for the general unrelated case, we present an exact algorithm which solves the simplified problem that the number of task types is restricted to $k(k \ll n)$. This problem is an extension of [8], and our algorithm runs in time $O(m \cdot n^{2(k-1)} \cdot \log Q)$, where Q is the difference between estimated values of upper and lower bounds for the optimum schedule length.

2. Scheduling on Two Processors

In this section we formulate the model for m=2 processors as 0-1 pro-

gramming problem. Based on the solution of its continuous relaxation, we develop a new ε-approximate algorithm. For the demonstration of its performance, numerical comparisons with [7] and [6] are presented.

2.1 Formulation and relaxation

Let x_i be the 0-1 variable which represents whether the task T_i is processed by processor $P_1(x_i=1)$ or by $P_2(x_i=0)$. Then, our problem to minimize the greater of $\sum_{i=1}^{n} \mu_{i1} \cdot x_i$ and $\sum_{i=1}^{n} \mu_{i2} \cdot (1-x_i)$ is expressed in the form of 0-1 programming as;

$$(2.1)\quad \begin{cases} \text{minimize} & f = y \\ \text{subject to} & \sum_{i=1}^{n} \mu_{i1} \cdot x_i \leqq y \\ & \sum_{i=1}^{n} \mu_{i2} \cdot (1-x_i) \leqq y \\ & y \geqq 0,\ x_i = 0 \text{ or } 1 \quad (i=1,2,\ldots,n) \ . \end{cases}$$

We consider the continuous relaxation of problem (2.1) where x_i is a real number in the interval [0 1], that means x_i portion of T_i is processed by P_1 and $(1-x_i)$ portion by P_2. Here we assume that the tasks have been sorted in the non-increasing order of μ_{i2}/μ_{i1} and reindexed so that

$$\mu_{12}/\mu_{11} \geqq \mu_{22}/\mu_{21} \geqq \ldots \geqq \mu_{n2}/\mu_{n1} \ .$$

We also assume, for the convenience of later discussions, that the processors become available (begin processing) at time $\alpha(P_1)$ and $\beta(P_2)$, where $\alpha, \beta \geqq 0$. Now by introducing slack variables $s_1, s_2, \lambda_1, \lambda_2, \ldots, \lambda_n$ and using M for $\sum_{i=1}^{n} \mu_{i2}$, (2.1) is rewritten as follows.

$$(2.2)\quad \begin{cases} \text{minimize} & f = y \\ \text{subject to} & \sum_{i=1}^{n} \mu_{i1} \cdot x_i - y + s_1 = -\alpha \\ & \sum_{i=1}^{n} \mu_{i2} \cdot x_i + y - s_2 = M+\beta \\ & x_i + \lambda_i = 1 \qquad (i=1,2,\ldots,n) \\ & x_i, y, s_1, s_2, \lambda_i \geqq 0 \end{cases}$$

Direct application of the primal simplex algorithm to (2.2) produces the next theorem.

Theorem 1. Let p be the task index such that

$$\sum_{i=1}^{p-1} \mu_{i1} + \alpha \leqq \sum_{i=p}^{n} \mu_{i2} + \beta \qquad \text{and}$$

$$\sum_{i=1}^{p} \mu_{i1} + \alpha > \sum_{i=p+1}^{n} \mu_{i2} + \beta$$

(if $\alpha + \sum_{i=1}^{n} \mu_{i1} \leqq \beta$, let p be $n+1$ or if $\alpha > \beta + \sum_{i=1}^{n} \mu_{i2}$, let p be 0),

then the optimum solution $\boldsymbol{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ for (2.2) is given by

$$x_i^* = \begin{cases} 1 & 1 \leqq i \leqq p-1 \\ \theta & i = p \\ 0 & p+1 \leqq i \leqq n \end{cases},$$

where $\theta = \dfrac{\beta + \sum_{i=p}^{n} \mu_{i2} - (\alpha + \sum_{i=1}^{p-1} \mu_{i1})}{\mu_{p1} + \mu_{p2}}$ $\quad (0 \leqq \theta < 1)$.

Proof: When $p = 0$ or $n+1$ the theorem is trivial, so it is sufficient to consider the case that $|\alpha - \beta| < \min(\sum_{i=1}^{n} \mu_{i1}, \sum_{i=1}^{n} \mu_{i2})$. In this case, (2.2) is solved by the simplex method as follows. At first, we eliminate the variable y from 1st and 2nd equations of (2.2) and rewrite it into a basic form with the basis $\{y, s_1, \lambda_1, \lambda_2, \ldots, \lambda_n\}$. Since $\alpha - \beta < M$ is the case, resultant tableau (a) of Table 2.1 is feasible. Starting with this, we repeat to select x_i as a new basic variable and eliminate λ_i from the basis for $i = 1, 2, \ldots, p-1$ to keep the solutions feasible. The final ($p-1$-th) tableau of this step is shown in (b), where $M - \alpha + \beta - \sum_{i=1}^{p-1} (\mu_{i1} + \mu_{i2}) = \beta + \sum_{i=p}^{n} \mu_{i2} - (\alpha + \sum_{i=1}^{p-1} \mu_{i1}) = B - A \geqq 0$. Next we introduce x_p into the basis. Since $\dfrac{B-A}{\mu_{p1} + \mu_{p2}} < 1$, the variable to be eliminated is s_1. At this time all entries of the first row in the resulting tableau (c) become nonpositive and the optimality condition is satisfied. The solution given by tableau (c) is what Theorem 1 asserts. □

Table 2.1 Simplex Tableaux

(a)

f	x_1	x_2	...	x_n	y	s_1	s_2	λ_1	λ_2	...	λ_n	constant
1	μ_{12}	μ_{22}	...	μ_{n2}	0	0	-1	0	0	...	0	M+β
0	w_1	w_2	...	w_n	0	1	-1	0	0	...	0	M-α+β
0	μ_{12}	μ_{22}	...	μ_{n2}	1	0	-1	0	0	...	0	M+β
0	1	0	...	0	0	0	0	1	0	...	0	1
0	0	1	...	0	0	0	0	0	1	...	0	1
.	.	.	.1.	.	.	.	.	.	.	.1.	.	.
0	0	0	...	1	0	0	0	0	0	...	1	1

$$M = \sum_{i=1}^{n} \mu_{i2} \cdot \qquad w_i = \mu_{i1} + \mu_{i2} \cdot$$

(b)

f	x_1	...	x_{p-1}	x_p	...	x_n	y	s_1	s_2	λ_1	...	λ_{p-1}	λ_p	...	λ_n	constant
1	0	...	0	μ_{p2}	...	μ_{n2}	0	0	-1	$-\mu_{12}$	...	$-\mu_{p-1\,2}$	0	...	0	B
0	0	...	0	w_p	...	w_n	0	1	-1	$-w_1$	...	$-w_{p-1}$	0	...	0	B-A
0	0	...	0	μ_{p2}	...	μ_{n2}	1	0	-1	$-\mu_{12}$	...	$-\mu_{p-1\,2}$	0	...	0	B
0	1	...	0	0	...	0	0	0	0	1	...	0	0	...	0	1
0	.	.1.	.	.	...	.	.	.	.	.	.1.	.	.	...	.	.
0	0	...	1	0	...	0	0	0	0	0	...	1	0	...	0	1
0	0	...	0	1	...	0	0	0	0	0	...	0	1	...	0	1
.	.	...	.	.	.1.	.	.	.	.	.	...	.	.	.1.	.	.
0	0	...	0	0	...	1	0	0	0	0	...	0	0	...	1	1

$$A = \alpha + \sum_{i=1}^{p-1} \mu_{i1} \cdot \qquad B = \beta + \sum_{i=p}^{n} \mu_{i2} \cdot$$

(c)

f	x_1	...	x_{p-1}	x_p	x_{p+1}	.	x_k	.	x_n	y	s_1	s_2	λ_1	.	λ_h	.	λ_{p-1}	λ_p	.	λ_n	constant
1	0	...	0	0	$\phi_p(p+1)$	.	$\phi_p(k)$	.	$\phi_p(n)$	0	$-\mu_{p2}/w_p$	$-\mu_{p1}/w_p$	$-\phi_p(1)$	.	$-\phi_p(h)$	.	$-\phi_p(p-1)$	0	.	0	η
0	0	...	0	1	w_{p+1}/w_p	.	w_k/w_p	.	w_n/w_p	0	$1/w_p$	$-1/w_p$	$-w_1/w_p$	.	$-w_h/w_p$	.	$-w_{p-1}/w_p$	0	.	0	θ
0	0	...	0	0	$\phi(p+1)$	.	$\phi_p(k)$	.	$\phi_p(n)$	1	$-\mu_{p2}/w_p$	$-\mu_{p1}/w_p$	$-\phi_p(1)$	.	$-\phi_p(h)$		$-\phi_p(p-1)$	0	.	0	η
0	1	...	0	0	0	.	0	.	0	0	0	0	0	.	0	.	0	0	.	0	1
.	.	.	1.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	...	1	0	0	.	0	.	0	0	0	0	0	.	0	.	0	.	.	.	1
0	0	...	0	0	$-w_{p+1}/w_p$	.	$-w_k/w_p$	.	$-w_n/w_p$	0	$-1/w_p$	$1/w_p$	w_1/w_p	.	w_h/w_p	.	w_{p-1}/w_p	1	.	0	$1-\theta$
0	0	...	0	0	0	.	0	.	0	0	0	0	0	.	0	.	0	0	1	0	1
.	.	...	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.
0	0	...	0	0	0	.	0	.	0	0	0	0	0	.	0	.	0	0	.	1	1

$$\phi_p(i) = \mu_{p1}\mu_{i1}(\mu_{i2}/\mu_{i1} - \mu_{p2}/\mu_{p1})/w_p \,. \qquad \theta = (B-A)/w_p = \frac{\beta + \sum_{i=p}^{n} \mu_{i2} - (\alpha + \sum_{i=1}^{p-1} \mu_{i1})}{\mu_{p1} + \mu_{p2}} \quad (0 \leq \theta < 1) \,.$$

$$\eta = \frac{\mu_{p1}(\beta + \sum_{i=p}^{n} \mu_{i2}) + \mu_{p2}(\alpha + \sum_{i=1}^{p-1} \mu_{i1})}{\mu_{p1} + \mu_{p2}} \,.$$

Rounding x_p^*, the solution for relaxed problem given in Theorem 1, to 1 or 0, i.e. shifting one of the fractions of task T_p to another processor, we can obtain two feasible schedules whose lengths are $\bar{f}_1(x_p=1)$ and $\bar{f}_2(x_p=0)$. Let $\bar{f}$ be the smaller of $\bar{f}_1$ and $\bar{f}_2$, f^p be the length of the schedule for $\mathcal{T} - \{T_p\}$ defined by x_i^* $(i=1,2,\ldots,p-1,p+1,\ldots,n)$, f^o be the optimum schedule length of the 0-1 problem (2.1) and f^* be the optimum value for the relaxed problem (2.2) respectively (see Fig. 1.).

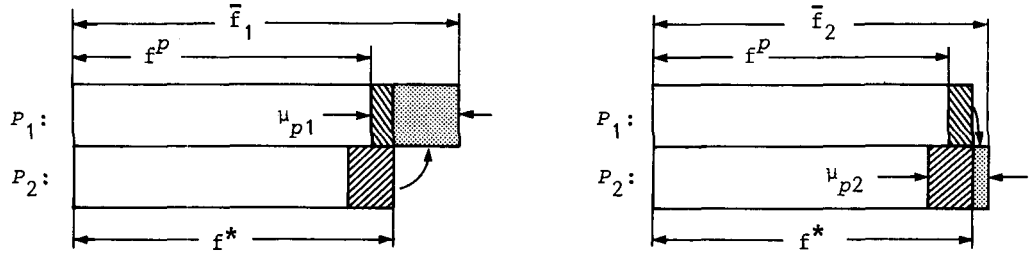

Fig. 1 Rounding of Task T_p

Then we have the next inequality.

Theorem 2. $\bar{f} < f^o + \min(\mu_{p1},\mu_{p2})$

Proof: Obviously $f^p < f^*$ and generally $f^* \leqq f^o$, therefore $f^p < f^o$. By definitions $f^p = \max(\bar{f}_1-\mu_{p1},\bar{f}_2-\mu_{p2})$, then $\bar{f}_1 \leqq f^p+\mu_{p1}$ and $\bar{f}_2 \leqq f^p+\mu_{p2}$. Accordingly,

$$\bar{f} = \min(\bar{f}_1,\bar{f}_2) \leqq \min(f^p+\mu_{p1},f^p+\mu_{p2}) = f^p + \min(\mu_{p1},\mu_{p2})$$
$$< f^o + \min(\mu_{p1},\mu_{p2}). \quad \square$$

2.2 Partially enumerative algorithm

The basic idea of our algorithm is that if $\min(\mu_{i1},\mu_{i2}) < f^o\cdot\varepsilon$ for all $1\leqq i\leqq n$ then Theorem 2 guarantees $\bar{f}$ being less than $f^o + f^o\cdot\varepsilon$ i.e. the relative error $\bar{f}/f^o$ being bounded by any given $\varepsilon>0$. To make use of this fact, using $\underline{f} = \sum_{i=1}^{n} \min(\mu_{i1},\mu_{i2})/2$ as the lower estimate for f^o, we divide the set $\mathcal{T}$ of tasks into two disjoint subsets;

$$\mathcal{T}_1 = \{T_k \mid \min(\mu_{k1},\mu_{k2}) \geqq \underline{f}\cdot\varepsilon\}$$
$$\mathcal{T}_2 = \{T_k \mid \min(\mu_{k1},\mu_{k2}) < \underline{f}\cdot\varepsilon\} = \mathcal{T} - \mathcal{T}_1.$$

Here we have the next theorem.

Theorem 3. $|\mathcal{T}_1| \leq \lceil 2/\varepsilon \rceil$

Proof: In the case of $q = \lceil 2/\varepsilon \rceil \geq n$ the theorem is trivial, hence we consider only the case of $q < n$. Sorting the tasks of $\mathcal{T}$ in nonincreasing order of $\min(\mu_{\cdot 1}, \mu_{\cdot 2})$, we have the sequence, $\min(\mu_{s_1 1}, \mu_{s_1 2}) \geq \ldots \geq \min(\mu_{s_q 1}, \mu_{s_q 2}) \geq \min(\mu_{s_{q+1} 1}, \mu_{s_{q+1} 2}) \geq \ldots \geq \min(\mu_{s_n 1}, \mu_{s_n 2})$. If $\min(\mu_{s_{q+1} 1}, \mu_{s_{q+1} 2}) \geq \underline{f} \cdot \varepsilon$ then we have the contradiction as follows;

$$2 \cdot \underline{f} = \sum_{i=1}^{n} \min(\mu_{s_i 1}, \mu_{s_i 2}) \geq \sum_{i=1}^{q+1} \min(\mu_{s_i 1}, \mu_{s_i 2}) \geq (q+1) \cdot \min(\mu_{s_{q+1} 1}, \mu_{s_{q+1} 2})$$

$$\geq (q+1) \cdot \underline{f} \cdot \varepsilon \geq (\lceil 2/\varepsilon \rceil + 1) \cdot \underline{f} \cdot \varepsilon) \geq (2/\varepsilon + 1) \cdot \underline{f} \cdot \varepsilon > 2 \cdot \underline{f} \ .$$

Therefore $\min(\mu_{s_{q+1} 1}, \mu_{s_{q+1} 2}) < \underline{f} \cdot \varepsilon$ and this means $|\mathcal{T}_1| \leq q = \lceil 2/\varepsilon \rceil$. □

The algorithm enumerates all the subschedules of $\mathcal{T}_1$ with finishing times denoted by α(for P_1) and $\beta(P_2)$. For each (α, β) it adds the subschedule of $\mathcal{T}_2$ determined by Theorem 1, and selects the total schedule of the minimum length (see Fig. 2).

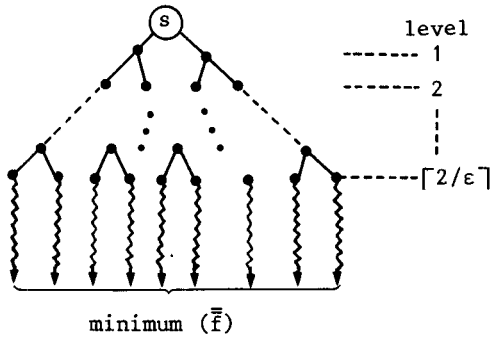

Fig. 2 Partial Enumeration

The formal description of the algorithm follows.

ALGORITHM A1: ($n, \mu_{11}, \ldots, \mu_{n1}, \mu_{12}, \ldots, \mu_{n2}$ are given)

step 1 (* preparations *)

calculate $\underline{f} = \sum_{i=1}^{n} \min(\mu_{i1}, \mu_{i2})/2$

divide $\mathcal{T}$ into $\mathcal{T}_1 \cup \mathcal{T}_2$ (* let $|\mathcal{T}_2|$ be r *)

sort tasks of $\mathcal{T}_2$ in nonincreasing order of $\mu_{\cdot 2}/\mu_{\cdot 1}$
(* suppose that $\mathcal{T}_2 = \{T_{s_1}, T_{s_2}, \ldots, T_{s_r}\}$ *)

let *MIN* be $+\infty$ (* current minimum schedule length *)

step 2 (* enumeration, addition, selection *)

for all subschedules of $\mathcal{T}_1$ (* α, β, x_i for $T_i \in \mathcal{T}_1$ are returned *)
do [

(* X,Y: current finishing times of P_1, P_2 *)

$X \leftarrow \alpha$; $Y \leftarrow \beta + \sum_{i \in \mathcal{T}_2} \mu_{i2}$; $j \leftarrow 1$;

while ($\max(X + \mu_{s_j 1}, Y - \mu_{s_j 2}) < Y$) **and** ($j \leqq r$) **do**
[$X \leftarrow X + \mu_{s_j 1}$; $Y \leftarrow Y - \mu_{s_j 2}$; $x_{s_j} \leftarrow 1$; $j \leftarrow j+1$;]

if $Y <$ *MIN* **then** [*MIN* $\leftarrow Y$; **save x**]

]

In step 2 each subschedule of $\mathcal{T}_1$ (specified by α,β and x_i for $T_i \in \mathcal{T}_1$), combined with additional schedule of $\mathcal{T}_2$ determined by Theorem 1, produces a total schedule with finishing time $\bar{\bar{f}}_{\alpha\beta}$. Let $f^o_{\alpha\beta}$ is the minimum schedule length under the condition that $\mathcal{T}_1$ has been scheduled with the finishing times $\alpha(P_1)$ and $\beta(P_2)$. Here $\bar{\bar{f}}_{\alpha\beta}$ is guaranteed to be less than $f^o_{\alpha\beta} + \underline{f}\cdot\varepsilon$ by Theorem 2 and the property of $\mathcal{T}_2$ that $\min(\mu_{\cdot 1}, \mu_{\cdot 2})$ of its any element is less than $\underline{f}\cdot\varepsilon$. After repeated comparisons, at the end of step 2 we obtain $\bar{\bar{f}} = \min_{\alpha,\beta} \bar{\bar{f}}_{\alpha\beta}$. Now we assume that the restriction of the overall optimum schedule (length f^o) to $\mathcal{T}_1$ gives the subschedule with finishing times $\alpha^*(P_1)$ and $\beta^*(P_2)$. This means that $f^o_{\alpha^*\beta^*} \leqq f^o$, while, f^o being optimum, $f^o \leqq f^o_{\alpha^*\beta^*}$ holds. Accordingly $f^o = f^o_{\alpha^*\beta^*}$ and obviously $\bar{\bar{f}} \leqq \bar{\bar{f}}_{\alpha^*\beta^*}$, then we have $\bar{\bar{f}} \leqq \bar{\bar{f}}_{\alpha^*\beta^*} \leqq f^o_{\alpha^*\beta^*} + \underline{f}\cdot\varepsilon \leqq f^o + f^o\cdot\varepsilon$, i.e. $\bar{\bar{f}}/f^o \leqq 1+\varepsilon$.

We now consider the time requirement of Algorithm A1. It is easily seen that step 1 takes at most $O(n\log n)$ time and that step 2 takes $(2^{|\mathcal{T}_1|}\cdot n)$ time. Let ε be, for example, 0.1 then step 2 executes its **for all** loop at most $2^{\lceil 2/\varepsilon \rceil} = 1000000$ times. This amount of computation is not so excessive for today's ordinary computers. Moreover it should be noted that $2^{\lceil 2/\varepsilon \rceil}$ is a constant on n, i.e. however large n becomes, this coefficient never changes. After all from the viewpoint of order in n, we can conclude that Algorithm A1 needs $O(n\log n)$ time for fixed ε.

The time requirement of A1 can be improved by eliminating the subschedule of step 2 both α and β of which are longer than those of already appeared subschedules, and by using the solution of the relaxed problem (f^*) as $\underline{f}$ to decrease $|\mathcal{T}_1|$. Thus improved algorithm is refered as A2. On the other hand the schedule length given by A1 may be slightly shortened, at cost of running time, by reassigning the tasks on the processor having the longer finishing time to another as long as the schedule length can be reduced.

2.3 Numerical experiment and comparison

In the following, results of the numerical experiment to examine time requirements of 4 algorithms, A1(ε=0.1), A2(revised A1, ε=0.1), I·K($\delta\cong$0.6) by Ibarra and Kim [7] and H·S(ε=0.1) by Horowitz and Sahni [6], are presented. We observe their running times for two types (case 1 and case 2) of data, and discuss their performances. The results in case 1 where μ_{ij}s are uniformly distributed random integers in the interval [1,100] are shown by Fig. 3, while those in case 2 where $\min(\mu_{i1},\mu_{i2})$ = constant(=33) for all i and the others are uniformly distributed in [34,83] are shown by Fig. 4. In both cases problem size n is moved from 10 to 150 with the certain interval, and for respective ns the averages of 25 trials (deviations are very small) are plotted. The absolute times measured (we used a medium scale computer HITAC M180 with speed of about 3 MIPS), of course, vary with the machine on which algorithms are executed, but the tendency (or ratio of values) is considered not to vary so much.

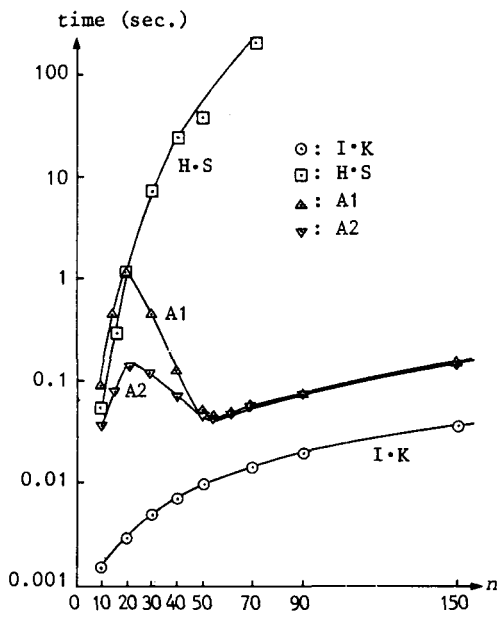

Fig. 3 Time Requirements in Case 1

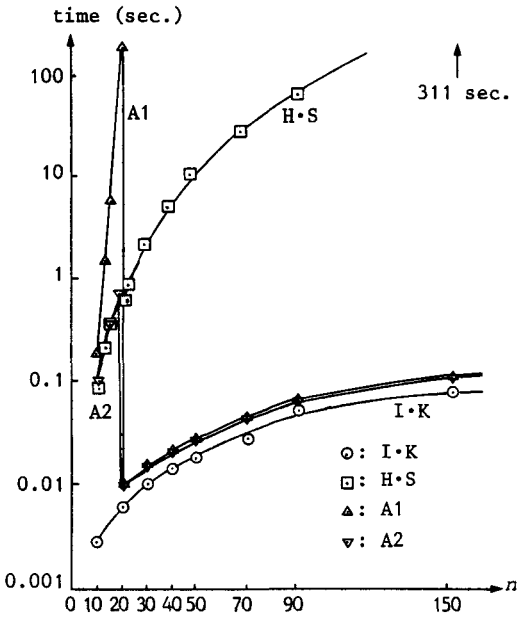

Fig. 4 Time Requirement in Case 2

In Fig. 3 the running time of I·K appears to follow the theoretical estimation $O(n\log n)$, and that of H·S to grow very rapidly. Algorithm H·S divides the interval $[1,10^{r+1}]$ into $n\cdot 10^{l+1}$ equal subintervals of size

$\lceil 10^{r-l}/n \rceil$ and executes dynamic programming type computation regarding integers in a same subinterval as identical, where $10^r < f^* < 10^{l+1}$ and $\varepsilon > 10^{-l}$. However, in our experiment, the size of subinterval is observed to be less than 2 or so. The value of $\varepsilon = 0.1$ seems to be too small for the above idea to work effectively. As for A1 and A2 their running times first increase and then decrease at the left side of n=50 or so where step 2 of algorithms dominates the total running time and $|\mathcal{T}_1|$ is maximized at about =20, while at the right side of it, they behave similarly to I·K, because $O(n\log n)$ time step 1 dominates the total running time. The difference between A1 and A2 shows that our improvement mentioned at the end of 2.2 is effective.

Fig. 4 shows the case that is most unfavorable to algorithm A1 or A2. For n<20 the running time of A1 appears to grow exponentially because $|\mathcal{T}_1|=n$ and step 2 dominates the total time, while that of A2 increase more slowly (at the same rate as H·S). For $n \geqq 20$ where $|\mathcal{T}_1|$ is zero, running times of A1 and A2 coincide and behave similarly to I·K. As for I·K and H·S, their running times are respectively almost same as in case 1. Through Fig. 3 and 4, I·K is seen to be the fastest of 4 algorithms.

Next, we consider the lengths of generated schedules. In the above case those of I·K, being far smaller than its theoretical bound $f^* \cdot \frac{1+\sqrt{5}}{2}$, are only a little longer than those of A1, A2 or H·S, but they are not always so. For data like

	1	2	3	4	5	6	7	8	9	10
μ_{i1}	22	52	89	10	75	64	70	800	76	74
μ_{i2}	35	82	140	17	118	101	110	1250	120	117

it generates the schedule with length 1250, while H·S and A2(A1) generate respective schedules with length 822 and 823 for the same data. The data unfavorable to I·K like this can be systematically produced, in other words, occur with certain probability.

Here we can conclude that algorithm A2 based on A1 is much faster than H·S under the same condition of accuracy and more accurate than I·K at the moderate cost of running time.

3. Scheduling on m Processors

In this section we consider the case of m>2 processors by assuming that the execution times list of each task, $(\mu_{\cdot 1}, \mu_{\cdot 2}, \ldots, \mu_{\cdot m})$, is drawn from an arbitrary set of k different lists (task types). This restriction is a natural extension of the idea introduced in [8], and reflects the situation that we

have the large number of tasks to be scheduled, while most of them have the same suitability to processors.

In the following, we show that the above simplified problem can be solved by a polynomial time (in m and n for fixed k) algorithm. First we construct the algorithm for the special case that there are only two task types, and then generalize it to the case of k different task types.

3.1 The algorithm for two task types

Suppose that we are given N_1 tasks of type 1 having the execution times list $(\mu_{11},\mu_{12},\ldots,\mu_{1m})$ and N_2 tasks of type 2 with the list $(\mu_{21},\mu_{22},\ldots,\mu_{2m})$, where $N_1+N_2 = n$. Let us consider the following decision problem: For a given positive integer D, is there a schedule of these n tasks on processors $P_1,P_2,\ldots,P_m$ whose length is less than or equal to D? If there is an algorithm to olve this decision problem, by binary search method using this algorithm as a test procedure, we can find the minimum value of D for which the decision problem is feasible, i.e. the optimum schedule length. The search process terminates because $\mu_{..}$s are all integral. If the above decision problem can be solved in time T, then the overall algorithm will run in time $T\cdot\log(P-p)$, where $P = \lceil n/m\rceil\cdot\max_{i,j}(\mu_{ij})$ and $p = \lceil n/m\rceil\cdot\min_{i,j}(\mu_{ij})$ are the initial upper and lower bounds of the optimum schedule length. Our problem is now reduced to solving the decision problem in polynomial time.

Let $x_{1j}(x_{2j})$ be the number of type 1 (type 2) tasks which are scheduled on processor $P_j(j=1,2,\ldots,m)$. Then, the decision problem is expressed as follows: For given D, are there nonnegative integers $x_{1j},x_{2j}(j=1,2,\ldots,m)$ such that

$$(3.1)\quad \begin{cases} \mu_{1j}\cdot x_{1j} + \mu_{2j}\cdot x_{2j} \leqq D & (j=1,2,\ldots,m) \\ \sum_{j=1}^{m} x_{1j} \geqq N_1 \\ \sum_{j=1}^{m} x_{2j} \geqq N_2 \ ? \end{cases}$$

Regarding each pair (x_{1j},x_{2j}) as a point on X_1-X_2 plane, we can interpret the first m inequalities of (3.1) as the condition that the nonnegative integer valued point (x_{1j},x_{2j}) must locate on or below the straight line $\mu_{1j}\cdot x_{1j} + \mu_{2j}\cdot x_{2j} = D$ for $j=1,2,\ldots,m$ (see Fig. 5).

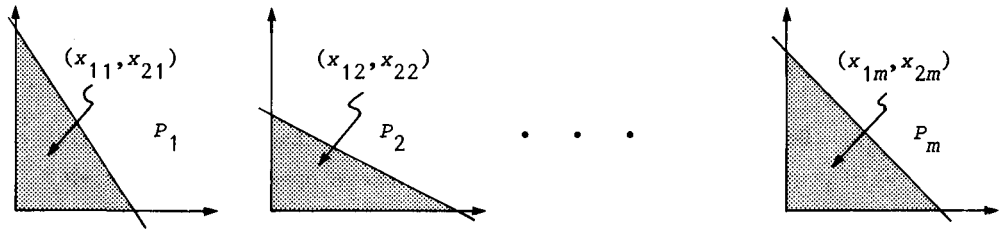

Fig. 5 Candidate Regions for Formula (3.1)

Thus the problem is changed to pick m points, one point from each region, so that the sum of the first coodinate of the m points is at least N_1 and the sum of the second coodinate is at least N_2. In choosing these points, it is enough to consider only "boundary point" of each region, the points whose second coodinate are the largest for each (fixed) first coodinate, 0,1,2,..., $\min(n, \lfloor D/\mu_{1j} \rfloor)$, since every other point is subsumed by one of them.

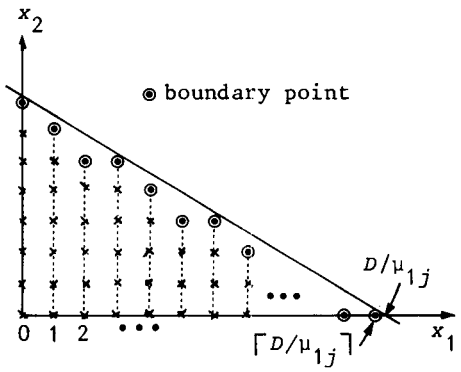

Fig. 6 Boundary Points for P_j

The number of boundary points of each region is $O(n)$ ($\leq n+1$). So simple enumeration of m points, choosing one points from each boundary points, requires $O(n^m)$ time. However, as shown in the following, it is possible to perform substantially same work in polynomial time.

First we consider the subsystem of two processors, say P_1 and P_2, denoted by $\langle P_1\text{-}P_2 \rangle$. Combining each boundary point of P_1 and each of P_2 (respectively (b_{11}, b_{21}) and (b_{12}, b_{22})), we compute all composite points $(b_{11}+b_{12}, b_{21}+b_{22})$. Among $O(n^2)$ these composite points, only those the second components of which are the largest for respective values of the first component are retained since the other composite points are subsumed by them. The number of these

"composite boundary points" is $O(n)$ ($\leq n+1$). Next, combining each of the composite boundary points of $<P_1\text{-}P_2>$ and each boundary point of P_3, (respectively $(b_1^{(12)}, b_2^{(12)})$ and (b_{13}, b_{23})), we compute all composite points $(b_1^{(12)} + b_{13}, b_2^{(12)} + b_{23})$. Here, again, from $O(n^2)$ computed points we choose $O(n)$ points of $<P_1\text{-}P_3>$ processors system. Repeating these operations of $<P_1\text{-}P_j>$ processors system ($j=3,4,\ldots$), at the end we have the composite boundary points for total processors system. By examining the value of the second component of the point whose first component is N_1, we can determine the feasibility of our decision problem. It it is less than N_2 then x_{1j} and x_{2j} ($j=1,2,\ldots,m$) to satisfy (3.1) for given D do not exist else do exist.

For the formal description of the above procedure based on the dynamic programming principle, we introduce two series of arrays with $n+1$ entries $y^{(j)}$ and $u^{(j)}$ ($j=1,2,\ldots,m$) and the binary operation $\otimes$ between them. Each entry of $y^{(j)}$ is computed as

$$(3.2) \qquad y^{(j)}[t] = \begin{cases} \lfloor (D - t\cdot\mu_{1j})/\mu_{2j} \rfloor & 0 \leq t \leq \lfloor D/\mu_{1j} \rfloor \\ -\infty & \lfloor D/\mu_{1j} \rfloor < t \leq n \ , \end{cases}$$

that is, $(t, y^{(j)}[t])$ is a boundary point of P_j. We define $\otimes$ between two such arrays by

$$a \otimes b[t] = \max_{0\leq s\leq t} (a[s] + b[t-s]) \qquad (t=1,2,\ldots,n).$$

Later, we refer the index s^* ($0 \leq s^* \leq t$) that realizes the right-hand side (maximizes $a[s]+b[t-s]$) as "realization index" of $a \otimes b$. Now, $u^{(j)}$, which represents the composite boundary points for $<P_1\text{-}P_j>$ processors system, can be computed from $y^{(j)}$ and $u^{(j-1)}$ as

$$(3.3) \qquad u^{(j)} = y^{(j)} \otimes u^{(j-1)} \qquad (j=1,2,\ldots,m).$$

Initial array $y^{(0)}$ which works as a unit element in this operation is given by

$$u^{(0)}[t] = \begin{cases} 0 & t=0 \\ -\infty & 1\leq t\leq n \ . \end{cases}$$

Note that $y^{(j)}[t]$ or $u^{(j)}[t]$ holds the maximum number of type 2 tasks that can be processed by P_j or $P_1 \sim P_j$ within time length D under the condition that t type 1 tasks have been assigned.

When given schedule length D is optimum, the actual schedule which realizes this D is needed. For this purpose it is sufficient to record the realization index of $u^{(j)}[t]$ in an array, say $w^{(j)}[t]$, at each step of computing $u^{(j)}[t] = y^{(j)} \otimes u^{(j-1)}[t]$ ($j=1,2,\ldots,m$; $t=0,1,2,\ldots,n$). If $u^{(j)}[t]$

results in $-\infty$ the index is not needed. After all $w^{(j)}$s are determined, we can construct the schedule by backward iteration

$$(3.4)\qquad \begin{cases} x_{1j} = w^{(j)}[N_1 - \sum_{s=j+1}^{m} x_{1s}] \\ x_{2j} = \lfloor (D - \mu_{1j} \cdot x_{1j})/\mu_{2j} \rfloor \end{cases} \qquad (j=m,m-1,\ldots,1).$$

The formal description of the algorithm (named A3) for the decision problem follows.

ALGORITHM A3: (D is given)

```
(* initialization *)
   let u[0] ← 0; u[1] ← -∞; u[2] ← -∞; ... u[n] ← -∞;
   for j:=1 to m do (* main for loop *) [
        (* computation of y *)
           R ← D;  a ← μ1j;  b ← μ2j;  t ← 0;
           while R ≧ 0 do [ y[t] ← ⌊R/b⌋; R ← R-a; t ← t+1 ]
           while t ≦ n do [ y[t] ← -∞; t ← t+1 ]
        (* computation of next u *)
           for t:=0 to n do [ u'[t] ← max_s(y[s] + u[t-s]); w(j)[t] ← s* ]
        (* u ← u' *)
           for t:=0 to n do [ u[t] ← u'[t] ]
   ]
   if u[N1] ≧ N2 then YES (* feasible *) else NO (* infeasible *)
```

The computations of u and $w^{(j)}$ (operation $\otimes$) may be carried out alternatively in the following manner, where $y[n+1]$ and $u[n+1]$ are assumed to hold the value $-\infty$ as sentinels.

```
(* computation of next u *)
   for t:=0 to n do [ u'[t] ← -∞; ];  s ← 0;
   while y[s] ≧ 0 do [ r ← 0; t ← s;
        while u[r] ≧ 0 do [
           if u[r]+y[s] > u'[t] then [ u'[t] ← u[r]+y[s]; w(j)[t] ← s ]
           r ← r+1;  t ← t+1
        ]
        s ← s+1
   ]
```

Theoretical order of the time requirement does not change, but the actual running time will be shortend in most of cases.

3.2 Illustrative example

To illustrate the above algorithm A3, we consider the following example. Let m=3, k=2, N_1=4, N_2=7, $\mu_{1\cdot}$=(2 4 3) and $\mu_{2\cdot}$=(6 3 4). Suppose that 12 is given as D (it is the optimum schedule length in this case).

At first u is set to

$$(0, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty).$$

Next, y is computed at the first step of main **for**-loop as

$$y = (2, 1, 1, 1, 0, 0, 0, -\infty, -\infty, -\infty, -\infty, -\infty).$$

At the same time realization index

$$w^{(1)} = (0, 1, 2, 3, 4, 5, 6, -, -, -, -, -)$$

are recorded. In the next iteration of the main **for**-loop, we have

$$y = (4, 2, 1, 0, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty),$$

$$u = (6, 5, 5, 5, 4, 4, 4, 2, 1, 0, -\infty, -\infty),$$

and

$$w^{(2)} = (0, 0, 0, 0, 0, 0, 0, 1, 2, 3, -, -).$$

Here u represents composite boundary points for $P_1 \sim P_2$. At the third iteration of main **for**-loop we have

$$y = (3, 2, 1, 0, 0, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty),$$

$$u = (9, 8, 8, 8, 7, 7, 7, 6, 5, 4, 4, 2),$$

and

$$w^{(3)} = (0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 4).$$

Since $u[N_1]$=7 and N_2=7, the decision problem of this example with D = 12 is feasible. As for the actual schedule that realizes this schedule length, trace back of index arrays $w^{(3)}$, $w^{(2)}$ and $w^{(1)}$ gives the task assignment (x_{11},x_{21}) = (4,0), (x_{12},x_{22}) = (0,4) and (x_{13},x_{23}) = (0,3).

3.3 The general algorithm

The ideas in section 3.1 can be generalized to $k > 2$ different task types. In this case we are given the number of type i tasks N_i, its execution times list $(\mu_{i1},\mu_{i2},\ldots,\mu_{im})$ for i=1,2,...,k (ΣN_i=n) and positive integer D. The problem is to determine if there are nonnegative integral vectors $(x_{11}, x_{12},\ldots,x_{1m})$, $(x_{21},x_{22},\ldots,x_{2m})$, ..., $(x_{k1},x_{k2},\ldots,x_{km})$ such that

$$\text{(3.5)} \qquad \begin{cases} \sum_{i=1}^{k} \mu_{ij}\cdot x_{ij} \leqq D & (j=1,2,\ldots,m) \\ \sum_{j=1}^{m} x_{ij} \geqq N_i & (i=1,2,\ldots,k), \end{cases}$$

where x_{ij} represents the number of type i tasks scheduled on processor P_j.

To solve this problem, we can use the same method as in section 3.1 by extending the boundary points on X_1-X_2 plane (Fig. 5) to those of candidate regions bounded by surfaces

$$\mu_{1j}\cdot x_{1j} + \mu_{2j}\cdot x_{2j} + \cdots + \mu_{kj}\cdot x_{kj} = D \qquad (j=1,2,\ldots,m)$$

in k dimensional space. Corresponding to this, instead of $y^{(j)}[\cdot]$ and $u^{(j)}[\cdot]$, k-1 dimensional arrays $\bar{y}^{(j)}[t_1,t_2,\ldots,t_{k-1}]$ and $\bar{u}^{(j)}[t_1,t_2,\ldots,t_{k-1}]$ are introduced (j=1,2,...,m), and they hold the maximum number of type k tasks that can be processed by P_j or $\langle P_1$-$P_j\rangle$ within time length D under the condition that t_i type i (i=1,2,...,k-1) tasks have been assigned. These values stored in so indexed entries are computed as

$$\bar{y}^{(j)}[t_1,t_2,\ldots,t_{k-1}] = \begin{cases} \lfloor * \rfloor & \text{if } * = (D - \sum_{i=1}^{k-1} \mu_{ij}\cdot t_i)/\mu_{kj} \geqq 0 \\ -\infty & \text{otherwise,} \end{cases} \tag{3.6}$$

and

$$\bar{u}^{(j)}[t_1,t_2,\ldots,t_{k-1}] = \max_{0\leqq s_i \leqq t_i} (\bar{y}^{(j)}[s_1,s_2,\ldots,s_{k-1}] + \bar{u}^{(j-1)}[t_1-s_1,t_2-s_2,\ldots,t_{k-1}-s_{k-1}]), \tag{3.7}$$

where $0\leqq t_i \leqq n$ for i=1,2,...,k-1. After m repetitive computations of (3.6) and (3.7), by comparing $\bar{u}^{(m)}[N_1,N_2,\ldots,N_{k-1}]$ with N_k, we can determine the feasibility of the schedule with length D. The actual schedule for the optimum value is constructed from the realization indexes $s_1^*,s_2^*,\ldots,s_{k-1}^*$ recorded in k-1 dimensional arrays $\bar{w}_i^{(j)}[t_1,t_2,\ldots,t_{k-1}]$ (i=1,2,...,k-1) at each step of computing (3.7).

In the above manner algorithm A3 is generalized to the case of $k > 2$. Generalized A3 (named A3g) is given in the form of a function returning "YES" or "NO".

```
FUNCTION A3g (D)
    let ū(0)[0,0,...,0] ← 0;  let all other entries of ū(0) be -∞;
    for j:=1 to m do (* main for loop *) [
       for all 0 ≦ t_i ≦ n do [
          compute ȳ(j)[t_1,t_2,...,t_{k-1}] (* according to formula (3.6) *)
          compute ū(j)[t_1,t_2,...,t_{k-1}] (* according to formula (3.7) *)
          for i:=1 to k-1 do [ store s_i* into w̄_i(j)[t_1,t_2,...,t_{k-1}] ]
       ]
    ]
    if ū(m)[N_1,N_2,...,N_{k-1}] ≧ N_k then return "YES" (* feasible *)
                                 else return "NO"  (* infeasible *)
```

Lastly, formula (3.4) constructing an actual schedule is also extended to the $k-1$ dimensional version as

$$(3.8)\quad \begin{cases} x_{ij} = \bar{w}_i^{(j)}\left[N_1 - \sum_{s=j+1}^{m} x_{1s},\ N_2 - \sum_{s=j+1}^{m} x_{2s},\ \ldots,\ N_{k-1} - \sum_{s=j+1}^{m} x_{k-1\,s}\right] \\ \qquad (i=1,2,\ldots,k-1) \\ x_{kj} = \lfloor (D - \mu_{ij}\cdot x_{ij})/\mu_{kj} \rfloor \end{cases}$$

$$(j=m,m-1,\ldots,1).$$

The overall general algorithm A4, which determines the optimum schedule length and constructs the actual schedule for it, can be written as follows.

ALGORITHM A4: (N_i and μ_{ij} are given)

step 0 $P \leftarrow \lceil n/m \rceil \cdot \max(\mu_{ij})$; $p \leftarrow \lceil n/m \rceil \cdot \min(\mu_{ij})$;

step 1 **if** $P = p$ **then goto** step 3

step 2 $D \leftarrow$ **round**($(P+p)/2$)
if A3g(D) = "YES" **then** $P \leftarrow D$ **else** $p \leftarrow D$
goto step 1

step 3 (* construction of the actual schedule *)
for i:=1 **to** $k-1$ **do** [$Z_i \leftarrow N_i$]
(* Z_i holds $N_i - \sum_{s=j+1}^{m} x_{is}$ in the following **for-downto** loop. *)
for j:=m **downto** 1 **do** [
for i:=1 **to** $k-1$ **do** [$x_{ij} \leftarrow u_i^{(j)}[Z_1, Z_2, \ldots, Z_{k-1}]$]
$x_{kj} \leftarrow \lfloor (D - \sum_{i=1}^{k-1} \mu_{ij}\cdot x_{ij})/\mu_{kj} \rfloor$
for i:=1 **to** $k-1$ **do** [$Z_i \leftarrow Z_i - x_{ij}$]
]

The algorithm starts with appropriate upper and lower bounds (P and p) of the optimum schedule length, and determines the optimum value by using A3g as a decision function of bisectional choices, and constructs the actual schedule according to formula (3.8).

Finally we shall estimate the time and space requirements of algorithm A4. A4 repeats the loop step 1 ∿ step 2 $O(\log Q)$ times, where $Q = P-p$, so the function A3g is called also $O(\log Q)$ times. For each call A3g computes n^{k-1} entries of $\bar{y}^{(j)}$ and $\bar{u}^{(j)}$ for $j=1,2,\ldots,m$. $O(n)$ arithmetic operations are needed for each entry of $\bar{y}^{(j)}$ and $O(n^{k-1})$ additions and comparisons for each entry of $\bar{u}^{(j)}$. At step 3, $O(m(k+m))$ operations and references to $\bar{w}_i^{(j)}$ are performed in order to determine x_{ij} ($i=1,2,\ldots,k$; $j=1,2,\ldots,m$). From these

accounts we can estimate that the total running time of algorithm A4 is

$$O(\log Q)\cdot\{m\cdot O(n\cdot n^{k-1}) + m\cdot O(n^{k-1}\cdot n^{k-1})\} + m\cdot O(k+n)$$
$$\sim O(\log Q\cdot mn^{2(k-1)}).$$

The space needed by A4 is easily seen to be $O(mkn^{k-1})$ mainly for $\bar{w}_i^{(j)}$. It is possible to reduce this requirement to $O(n^{k-1})$ by computing realization indexes dynamically with increase of the running time.

4. Conclusion

We have presented an ε-approximate algorithm for the case of m=2 processors. This algorithm runs in fairly practical time for ε=0.1 or so, while the existing ε-approximate algorithm works as almost exact (enumerative) one for the same degree of ε. We have also shown that the running time of our algorithm is only a little (by constant term) larger than those of existing δ-approximate ($\delta \cong 0.5\sim0.6$) algorithms for any n.

Basic idea of our algorithm (introduced in [9]) is to treat dominant variables enumeratively and the rest approximately (continuous relaxation and rounding). The similar approach is found in [10], where a relaxation problem is solved first and then all possible assignments of fractional variables are generated. But the classification of variables according to their importance is not considered there. Our simple idea seems to be applicable to some other combinatorial optimization problems.

For the case of m>2 processors, we simplified the problem and have presented an exact algorithm to solve it. The running time of presented algorithm is considered to be reasonable only when the number of different task types (k) is small. Improvement of the running time may be possible by tighter estimations of lower and upper bounds for the optimum schedule length or more efficient computation of $\bar{u}^{(j)}$s, but it cannot overcome the growth of the time requirement when k becomes larger. To investigate various modifications of the original problem is the subject for a future study.

Acknowledgements

The auther wishes to thank Mr. Takao Hanada of the University of Electro-Communications for his discussions. He also wishes to thank the referees for their helpful comments and suggestions.

References

[1] Bruno, J., Coffman, E.G.Jr., and Sethi, R.: Scheduling Independent Tasks to Reduce Mean Finishing Time. *Comm. ACM*, Vol.17, No.7 (1974), 382-387.

[2] Coffman, E.G.Jr., Garey, M.R., and Johnson, D.S.: An Application of Bin-packing to Multiprocessor Scheduling. *SIAM J. Comput.*, Vol.7, No.1 (1978), 1-17.

[3] Devis, E., and Jaffe, J.M.: Algorithms for Scheduling Tasks on Unrelated Processors. *J.ACM*, Vol.28, No.4 (1981), 721-736.

[4] Garey, M.R., and Johnson, D.S.: *Computers and Intractability —— A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[5] Graham, R.L.: Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.*, Vol.17, No.2 (1969), 416-429.

[6] Horowitz, E., and Sahni, S.: Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J.ACM*, Vol.23, No.2 (1976), 317-327.

[7] Ibarra, O.H., and Kim, C.E.: Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *J.ACM*, Vol.24, No.2 (1977), 280-289.

[8] Leung, J.Y-T.: On Scheduling independent Tasks with Restricted Execution Time. *Opns. Res.*, Vol.30, No.1 (1982), 163-171.

[9] Numata, K.: An Approximate Algorithm for Scheduling Independent Tasks on Unrelated Processors (in Japanese). *Proceedings of the 1984th Fall Conference of the Operations Research Society of Japan*, 77-78.

[10] Potts, C.N.: Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines. *Discrete Appl. Math.*, Vol.10 (1985), 155-164.

Kazumiti NUMATA: Department of Computer-Science and Information Mathematics, The University of Electro-Communications, 1-5-1, Chofu-ga-Oka, Chofu- Shi, Tokyo, 182, Japan.