Int. J. Appl. Math. and Comp. Sci., 1999, Vol.9, No.4, 939–954

MEAN SQUARED LOAD CRITERIA FOR SCHEDULING INDEPENDENT TASKS

GINTAUTAS DZEMYDA*

Results of this paper extend the set of criteria which characterize the scheduling quality as well as the set of possible scheduling strategies. A new view on the minimum makespan criterion is presented in terms of the mean squared load of processing units. This leads in turn to the development of new scheduling algorithms. The interaction between processes of minimizing the new criteria and the maximum finishing time (makespan of the schedule) was discovered. We show the possibility of minimizing the maximum finishing time by minimizing the new criteria that characterize the mean squared load of processing units. Moreover, the optimal workload of processing units determined with the use of the proposed criteria is usually smoother (more balanced) than that found for traditional ones.

Keywords: parallel processing, scheduling, combinatorial problems, minimum makespan problem, mean squared load

1. Introduction

The set of r independent tasks $T = \{T_1, \ldots, T_r\}$ has to be processed using p identical processing units (processors) P_i, $i = 1, \ldots, p$, $p < r$. Task T_j corresponds to the deterministic process which uses any single processor in a time μ_j. The processing of a task cannot be interrupted. Our aim is to find a schedule (an assignment of tasks to processors) which ensures a 'balanced workload' of processors. In this paper, we show that this generally formulated goal can be achieved using various scheduling criteria and thus it can be interpreted in several alternative ways. Next, we propose a new class of such criteria based on the mean squared load of processors which also creates a completely new class of scheduling problems. Finally, we discuss and investigate newly formulated problems as well as solution algorithms.

2. Minimum Makespan Problem

One of the early recognized criteria for a balanced processor workload is the makespan (maximum completion time). It tends to minimize the workload of the most loaded

* Institute of Mathematics and Informatics, 4, Akademijos St., 2600 Vilnius, Lithuania, e-mail: dzemyda@ktl.mii.lt

processor. For that purpose, let us denote

- by $A = (A_1, \ldots, A_p)$ a partition of the set T, where A_l is a group (set) of tasks scheduled for the l-th processing unit, and
- by $\mu(I) = \sum_{j \in I} \mu_i$ the sum of processing times for some subset $I \subseteq T$.

The makespan is given by

$$C_{\max}(A) = \max_{1 \le l \le p} \mu(A_l). \tag{1}$$

The problem of scheduling independent tasks on identical processors with criterion (1) is classical in scheduling theory (Hochbaum and Shmoys, 1987). It is also called the problem of minimizing the makespan of a schedule or the minimum makespan problem (Hochbaum and Shmoys, 1987). The problem is strongly NP-hard already for $p = 2$. Various computer architectures, ways of exploiting parallelism and the occurrence of applied problems influence the scheduling strategy (Blazewicz *et al.*, 1991; Brucker, 1998; Hochbaum and Shamir, 1990; Hochbaum and Landy, 1997; Hubsher and Glover, 1994; Lam, 1988; Lilja, 1991; Nawrocki *et al.*, 1998). Therefore, one can make out a schedule in several ways using different criteria. In what follows, we briefly summarize main results for the minimum makespan problem.

The classical largest-processing-time scheduling (LPT) (Graham, 1969) produces schedules which tend to maximize the mean finishing time at each point in the schedule, but minimize the maximum finishing time (1). The LPT strategy minimizing the maximum finishing time was proposed and investigated by Graham (1969): A free processor always starts executing the longest remaining unexecuted task. Denote by G such a scheduling algorithm. The scope of G is, in fact, to search for a new order of tasks T_i, $i = \overline{1,r}$, where they are arranged in decreasing order of their execution times. Therefore, the schedules produced by G have no empty groups of tasks: $A_l \neq \emptyset$, $l = \overline{1,p}$.

Graham's algorithm produces a schedule that has a makespan M at most $(4/3 - 1/3p)C^*_{\max}(A)$, where $C^*_{\max}(A)$ is the optimal value of $C_{\max}(A)$, i.e.

$$C^*_{\max}(A) \le M \le \left(\frac{4}{3} - \frac{1}{3p}\right) C^*_{\max}(A).$$

Other efficient algorithms for solving the minimum makespan problem are (a) MULTIFIT which delivers a schedule with makespan at most $1.22C^*_{\max}(A)$, $1.2C^*_{\max}(A)$, or even $(72/61)C^*_{\max}(A)$ (see (Hochbaum and Shmoys, 1987) for more details on these three cases), and (b) the dual approximation algorithm (Hochbaum and Shmoys, 1987), which produces a solution with makespan at most $(6/5 + 2^{-k})C^*_{\max}(A)$, where k is the number of iterations of binary search.

The advantages of Graham's algorithm are its simplicity and, as shown in further sections, that it produces a schedule that cannot be improved by transferring a single task from one group of tasks to another. MULTIFIT and dual approximation algorithms can find better schedules when compared with those found by G. These algorithms, however, are much more complex.

In this paper, we present a new view on the criterion of minimum makespan in terms of the mean squared load of processing units.

3. Mean Squared Load Scheduling Criteria

In the sequel, we propose new criteria providing a balanced workload of processors, which combine the deviation between the 'current' and an 'ideal' load of processors in a more sophisticated way. The 'ideal' load is defined by

$$\mu(A_1) = \cdots = \mu(A_p) = \bar{\mu} = \mu(T)/p, \tag{2}$$

where $\bar{\mu} = \mu(T)/p$ is the mean load of a processing unit. Equation (2) makes a basis for new scheduling criteria: we should tend to minimize the absolute values of the differences between all the pairs of the first p terms in (2).

We introduce three intuitively clear criteria that characterize the load of processing units from different standpoints:

(a) the mean sum of all the squared differences between all the pairs of the first p terms in (2):

$$C_1(A) = \frac{2}{p(p-1)} \sum_{l=1}^{p-1} \sum_{k=l+1}^{p} \left(\mu(A_k) - \mu(A_l)\right)^2, \tag{3}$$

(b) the variation of the load of processing units:

$$C_2(A) = \frac{1}{p} \sum_{l=1}^{p} \left(\mu(A_l) - \bar{\mu}\right)^2, \tag{4}$$

(c) the mean squared load of processing units:

$$C(A) = \frac{1}{p} \sum_{l=1}^{p} \mu^2(A_l). \tag{5}$$

It is necessary to minimize these criteria by seeking a best partition of tasks $T_1, \ldots, T_r$. The coefficient $2/p(p-1)$ introduced in (3) is related to the number of different combinations of indices l and k in (3), which is equal to $p(p-1)/2$.

As shown below, criteria $C_1(A)$ and $C_2(A)$ are directly related to $C(A)$.

Property 1. *Criteria $C(A)$, $C_1(A)$ and $C_2(A)$ are equivalent, i.e. if A^* is the optimal partition for $C(A)$, then A^* is the optimal partition for $C_1(A)$ (as well as for $C_2(A)$) and vice versa.*

Proof. The result becomes obvious if we make the following transformations in the expressions for $C_1(A)$ and $C_2(A)$:

$$\begin{aligned} C_1(A) &= \frac{1}{p(p-1)} \sum_{l=1}^{p} \sum_{k=1}^{p} \left(\mu(A_k) - \mu(A_l)\right)^2 \\ &= \frac{1}{p(p-1)} \left[p \sum_{k=1}^{p} \mu^2(A_k) - 2 \sum_{k=1}^{p} \mu(A_k) \sum_{l=1}^{p} \mu(A_l) + p \sum_{l=1}^{p} \mu^2(A_l) \right] \\ &= \frac{1}{p(p-1)} \left[2p \sum_{l=1}^{p} \mu^2(A_l) - 2 \left(\sum_{l=1}^{p} \mu(A_l) \right)^2 \right] = \frac{2p}{p-1} \left(C(A) - \bar{\mu}^2 \right), \end{aligned}$$

$$C_2(A) = \frac{1}{p}\sum_{l=1}^{p}\mu^2(A_l) - \frac{2}{p}\sum_{l=1}^{p}\mu(A_l)\bar{\mu} + \frac{1}{p}\sum_{l=1}^{p}\bar{\mu}^2$$

$$= C(A) - \frac{2\bar{\mu}}{p}\sum_{l=1}^{p}\mu(A_l) + \bar{\mu}^2 = C(A) - \bar{\mu}^2.$$

Consequently, we get expressions in which only $C(A)$ depends on the partition of tasks. This completes the proof. ■

From the transformations above in the expressions for $C_1(A)$ and $C_2(A)$, we obtain the following relation between the two criteria:

$$C_1(A) = \frac{2p}{p-1}C_2(A). \tag{6}$$

Property 2. *The problem of minimizing $C(A)$ is NP-complete already for $p = 2$.*

Proof. Instead of a full formal proof, we will outline its key elements only. To this aim, we refer to the known NP-complete problem

> 2PARTITION: Given a set of items $N = \{1, \ldots, n\}$ with sizes $s_1, \ldots, s_n$, respectively, where $\sum_{i=1}^{n} s_i = 2b$, does there exist a subset $I \subset N$ such that $\sum_{i\in I} s_i = b$?

and the decision version of our scheduling problem

> SCHEDULING: Given a set of tasks $T = \{1, \ldots, r\}$ with respective processing times $\mu_1, \ldots, \mu_r$, which should be processed on p processors, and a number y, does there exist a workload of processors $A = (A_1, \ldots, A_p)$ such that $C(A) \le y$?

We will show that 2PARTITION can be transformed to SCHEDULING in polynomial time using the transformation $p = 2$, $r = n$, $\mu_i = s_i$, $i = \overline{1,n}$, $y = 2b^2$.

Let us consider the solvability of both the problems:

2PARTITION $\rightarrow$ SCHEDULING: Assume that there exists a solution of the 2PARTITION problem. Then there exists a subset I such that $\sum_{i\in I} s_i = \sum_{i\in N\setminus I} s_i = b$. We construct a solution of the SCHEDULING problem putting $A = (A_1, A_2)$, $A_1 = I$, $A_2 = N\setminus I$. One can verify that $C(A) = \mu^2(I) + \mu^2(N\setminus I) = 2b^2$, which means that the required solution to SCHEDULING also exists.

SCHEDULING $\rightarrow$ 2PARTITION: Assume that there exists a solution to the SCHEDULING problem. Then there exists a workload $A = (A_1, A_2)$ such that $C(A) \le y$. Since $A_1 \cup A_2 = T$ and $\mu(T) = 2b$, we have $C(A) = \mu^2(A_1) + \mu^2(T\setminus A_1) = \mu^2(A_1) + [2b - \mu(A_1)]^2 \le y = 2b^2$. The last inequality has the unique solution $\mu(A_1) = b$. We construct a solution to the 2PARTITION problem putting $I = A_1$. One can verify that $\sum_{i\in I} s_i = b$, which means that the required solution to 2PARTITION also exists. This completes our proof. ■

Property 3. *The problems of minimizing* $C_{\max}(A)$ *and* $C(A)$ *for* $p = 2$ *are equivalent, i.e. if* A^* *is an optimal partition of* $C(A)$*, then* A^* *is optimal for* $C_{\max}(A)$ *and vice versa.*

Proof. Without loss of generality we may assume that $\mu(A_1) \geq \mu(A_2)$. Then $C_{\max}(A) = \mu(A_1)$ and, since $\mu(A_1) + \mu(A_2) = \mu(T)$, we have $\mu(A_1) \geq \mu(T)/2$ and

$$2C(A) = \mu^2(A_1) + \mu^2(A_2) = \mu^2(A_1) + \left[\mu(T) - \mu(A_1)\right]^2$$

$$= 2C_{\max}^2(A) - 2\mu(T)C_{\max}(A) + \mu^2(T).$$

The right-hand side of the last equality is non-decreasing for $\mu(A_1) \geq \mu(T)/2$, therefore minimizing $C_{\max}(A)$ we minimize $C(A)$. Conversely, the last transformation has a one-to-one inverse

$$C_{\max}(A) = \frac{1}{2}\left[\mu(T) + \sqrt{4C(A) - \mu^2(T)}\right]$$

for $\mu(A_1) \geq \mu(T)/2$ and therefore a similar conclusion can be drawn. This completes the proof. ■

The following example shows the non-equivalence of different loading criteria and superiority of $C(A)$ over $C_{\max}(A)$ for p greater than 2: $p = 3$, $r = 5$, $\mu_1 = \cdots = \mu_4 = 1$, $\mu_5 = 3$. Let us consider two different partitions $A = (A_1, \ldots, A_p)$ and $B = (B_1, \ldots, B_p)$: $A_1 = \{1\}$, $A_2 = \{2, 3, 4\}$, $A_3 = \{5\}$ and $B_1 = \{1, 2\}$, $B_2 = \{3, 4\}$, $B_3 = \{5\}$. In this case, $C_{\max}(A) = 5$, $C_{\max}(B) = 5$, $C(A) = 6.33$, $C(B) = 5.67$, $C_1(A) = 2.67$, $C_1(B) = 0.67$, $C_2(A) = 0.89$, and $C_2(B) = 0.22$. It is clear that $C_{\max}(A) = C_{\max}(B)$, but $C(A) > C(B)$, $C_1(A) > C_1(B)$ and $C_2(A) > C_2(B)$. Moreover, partition B is intuitively smoother (more balanced) than A in terms of the newly introduced criteria. Figure 1 illustrates graphically the workload of processing units from the standpoint of the number of scheduled tasks.

Criteria $C(A)$, $C_1(A)$ and $C_2(A)$ extend the set of possible scheduling strategies and allow us to create special scheduling algorithms. $C(A)$ is simpler when compared with $C_1(A)$ or $C_2(A)$. Therefore, we shall describe the minimization of $C(A)$ in more detail.

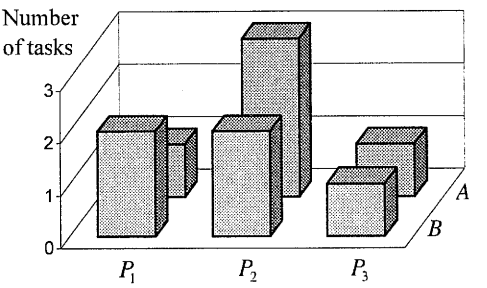

Fig. 1. The workload of processing units.

4. Mathematical Programming Formulations

Mathematical programming formulations for machine scheduling are surveyed in (Blazewicz *et al.*, 1991). In this section, we will follow the notation introduced therein. It is more complicated in comparison with that in other sections of our paper, but it makes it possible to formulate mathematical programming problems more precisely.

Problem (1) can be formulated as follows (Blazewicz *et al.*, 1991):

$$\text{Minimize } Y \tag{7}$$

subject to

$$\sum_{k=1}^{r}\sum_{i=1}^{p} x_{ij}^{k} = 1 \quad (j = \overline{1,r}), \tag{8}$$

$$\sum_{j=1}^{r} x_{ij}^{k} \le 1 \quad (k = \overline{1,r};\ i = \overline{1,p}), \tag{9}$$

$$\sum_{k=1}^{r}\sum_{j=1}^{r} \mu_j x_{ij}^{k} \le Y \quad (i = \overline{1,p}), \tag{10}$$

$$x_{ij}^{k} \in \{0,1\} \quad (j,k = \overline{1,r};\ i = \overline{1,p}). \tag{11}$$

Here x_{ij}^{k} is the decision variable which takes values 1 or 0: $x_{ij}^{k} = 1$ if job j is the k-th job processed by processing unit i, and 0 otherwise. Denote by $M1$ problem (7)–(11).

The problem of optimization of $C(A)$, $C_1(A)$ and $C_2(A)$ (i.e. the problem of minimizing the mean squared load of processing units) may be formulated as follows:

$$\text{Minimize } \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^{k}\right)^2 \tag{12}$$

subject to

$$\sum_{k=1}^{r}\sum_{i=1}^{p} x_{ij}^{k} = 1 \quad (j = \overline{1,r}), \tag{13}$$

$$\sum_{j=1}^{r} x_{ij}^{k} \le 1 \quad (k = \overline{1,r};\ i = \overline{1,p}) \tag{14}$$

$$x_{ij}^{k} \in \{0,1\} \quad (j,k = \overline{1,r};\ i = \overline{1,p}). \tag{15}$$

Denote by $M2$ problem (12)–(15). All the functions in Problem $M1$ (both the objective function and constraints) are linear. Problem $M2$ has linear constraints but its objective function is nonlinear.

In general, Problems $M1$ and $M2$ have different optimal values of decision variables x_{ij}^k. However, let us look for similarities of these two problems and for common strategies of optimization.

Rewriting the system of inequalities (10), we obtain

$$\left(\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k\right)^2 \le Y^2 \quad (i=\overline{1,p}). \tag{16}$$

This does not change the solution to Problem $M1$.

Let us sum up all the p inequalities of (16) and divide the result by p:

$$\frac{1}{p}\sum_{i=1}^{p}\left(\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k\right)^2 \le Y^2. \tag{17}$$

Inequality (17) relates the objective functions of Problems $M1$ and $M2$: On the left-hand side of inequality (17) we have the objective function of Problem $M2$, and on the right-hand side we have the squared objective function of Problem $M1$. A solution to Problem $M1$ should satisfy this inequality. Therefore, the global solution $C^*_{\max}(A)$ to Problem $M1$ satisfies the condition

$$\frac{1}{p}\sum_{i=1}^{p}\left(\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k\right)^2 \le \left[C^*_{\max}(A)\right]^2 \tag{18}$$

because, in this case,

$$\max_{1\le i\le p}\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k = C^*_{\max}(A)$$

and

$$\max_{1\le i\le p}\left(\sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k\right)^2 = \left[C^*_{\max}(A)\right]^2.$$

Remark 1. From (7), (10), (12), (17), and (18) it follows that if we tend to minimize the mean squared load of processing units $C(A)$, then we can expect a reduction in the makespan $C_{\max}(A)$. However, if we tend to minimize the makespan, then the mean squared load of processing units may grow (and vice versa). This can be shown by the following example. Let us consider $p=5$, $T=\{T_1,\dots,T_{14}\}$, $\mu_j=1$, $j=\overline{1,14}$, partition $A=(A_1,\dots,A_p)$: $A_1=\{1,2,3,4\}$, $A_2=\{5,6,7,8\}$, $A_3=\{9,10,11,12\}$, $A_4=\{13\}$, $A_5=\{14\}$, $C_{\max}(A)=4$, $C(A)=10$, and partition $B=(B_1,\dots,B_p)$: $B_1=\{1,2,3,4,5\}$, $B_2=\{6,7\}$, $B_3=\{8,9\}$, $B_4=\{10,11\}$, $B_5=\{12,13,14\}$, $C_{\max}(B)=5$, $C(B)=9.2$. It is clear that $C_{\max}(A)<C_{\max}(B)$, but $C(A)>C(B)$.

Remark 1 indicates a search direction for new strategies of reduction in the makespan $C_{\max}(A)$: minimization of $C_{\max}(A)$ via minimization of $C(A)$ ($C_1(A)$ or $C_2(A)$ as well, see Property 1).

Following the notation of the Introduction and Section 2, we get $\mu(A_i) = \sum_{k=1}^{r}\sum_{j=1}^{r}\mu_j x_{ij}^k$, $i = \overline{1,p}$. Therefore, from (7), (10), (12), and (17) we can deduce the following common strategy of solving Problems $M1$ and $M2$: At each step of the optimization algorithm, reduce the maximal squared sum from among $\mu^2(A_i)$ (and the maximal sum from among $\mu(A_i)$ as well), $i = \overline{1,p}$. Doing so, we reduce the values of $C_{\max}(A)$ and we can expect a reduction in $C(A)$ (naturally, in $C_1(A)$ and $C_2(A)$, too).

The following theorem makes it possible to determine the cases where the value of $C(A)$ is reduced. Let $A = (A_1, \ldots, A_p)$ be a given partition of tasks $T_1, \ldots, T_r$. Without loss of generality, assume that the groups are numbered so that $\mu(A_1) = \max_{1\le i\le p}\mu(A_i)$ and the number of tasks scheduled to the processing unit P_1 is greater than 1 (if this number is equal to 1, the minimum makespan has been found: it is equal to $\mu(A_1)$). Let $X = (X_1, \ldots, X_p)$ be a partition of the group A_1. In this case, we have $\mu(A_1) = \sum_{i=1}^{p}\mu(X_i)$ and so we will try to reduce the load of P_1 by passing subgroups X_i, $i = \overline{2,p}$ to units P_i, $i = \overline{2,p}$, respectively.

Theorem 1. *Let $A_1, \ldots, A_p$ be a partition of $T = (T_1, \ldots, T_r)$ such that $\mu(A_1) \ge \mu(A_i)$, $i = \overline{2,p}$, $X = (X_1, \ldots, X_p)$ be a partition of A_1, and $B = (B_1, \ldots, B_p)$ be the redistributed partition: $B_1 = X_1 = A_1 \setminus \bigcup_{i=2}^{p} X_i$, $B_i = A_i \cup X_i$, $i = \overline{2,p}$. Then:*

1. *$C(B) < C(A)$ if and only if*

$$\mu(A_1) > \frac{\sum\limits_{i=2}^{p}\mu(X_i)\left[\mu(A_i)+\mu(X_i)\right]}{\sum\limits_{i=2}^{p}\mu(X_i)} + \frac{\sum\limits_{i=2}^{p}\sum\limits_{\substack{j=2\\ j\ne i}}^{p}\mu(X_i)\mu(X_j)}{2\sum\limits_{i=2}^{p}\mu(X_i)}, \quad \text{and} \tag{19}$$

2. *If*

$$\mu(A_1) \ge \mu(A_i) + \mu(X_i) + \frac{1}{2}\sum_{\substack{j=2\\ j\ne i}}^{p}\mu(X_j), \quad i = \overline{2,p} \tag{20}$$

holds with at least one strict inequality, then $C(B) < C(A)$, i.e. the value of $C(A)$ will be reduced after the redistribution of tasks.

Proof. Cases 1 and 2 will be proved separately.

Case 1. Let the assumptions of Theorem 1 be satisfied. We shall look for conditions when $C(B) < C(A)$. It follows that

$$\begin{aligned} p[C(A) - C(B)] &= \mu^2(A_1) - \mu^2(B_1) + \sum_{i=2}^{p} \left[\mu^2(A_i) - \mu^2(B_i)\right] \\ &= \mu^2(A_1) - \left[\mu(A_1) - \sum_{i=2}^{p} \mu(X_i)\right]^2 \\ &\quad + \sum_{i=2}^{p} \mu^2(A_i) - \sum_{i=2}^{p} \left[\mu(A_i) + \mu(X_i)\right]^2 \\ &= 2\mu(A_1) \sum_{i=2}^{p} \mu(X_i) - \left[\sum_{i=2}^{p} \mu(X_i)\right]^2 \\ &\quad - 2\sum_{i=2}^{p} \mu(A_i)\mu(X_i) - \sum_{i=2}^{p} \mu^2(X_i) > 0. \end{aligned} \tag{21}$$

Hence

$$\begin{aligned} \mu(A_1) &> \frac{\left[\sum_{i=2}^{p} \mu(X_i)\right]^2 + 2\sum_{i=2}^{p} \mu(A_i)\mu(X_i) + \sum_{i=2}^{p} \mu^2(X_i)}{2\sum_{i=2}^{p} \mu(X_i)} \\ &= \frac{\sum_{i=2}^{p}\sum_{j=2}^{p} \mu(X_i)\mu(X_j) - \sum_{i=2}^{p} \mu^2(X_i) + 2\sum_{i=2}^{p} \mu(A_i)\mu(X_i) + 2\sum_{i=2}^{p} \mu^2(X_i)}{2\sum_{i=2}^{p} \mu(X_i)}. \end{aligned} \tag{22}$$

From (22) we get (19), which completes the proof of this case.

Case 2. Multiplying (22) by $2\sum_{i=2}^{p} \mu(X_i)$, after some transformations, we get

$$\sum_{i=2}^{p} \{2\mu(X_i)\mu(A_1)\} > \sum_{i=2}^{p} \left\{\mu(X_i)\sum_{j=2}^{p} \mu(X_j) + \mu^2(X_i) + 2\mu(A_i)\mu(X_i)\right\}. \tag{23}$$

Let us analyze the expressions in the brackets. If all $p-1$ expressions

$$\mu(A_1) - \left[\mu(A_i) + \mu(X_i) + \frac{1}{2}\sum_{\substack{j=2 \\ j\neq i}}^{p} \mu(X_j)\right], \quad i = \overline{2,p}$$

are nonnegative and at least one of them is greater than zero, then (19) is valid, i.e. the value of $C(A)$ will be reduced if the second proposition of Theorem 1 holds. This completes the proof of the theorem. ■

Equation (20) may be rewritten as follows:

$$\mu(A_1) \geq \mu(A_i) + \mu(X_i) + \frac{1}{2}\big[\mu(A_1) - \mu(X_1) - \mu(X_i)\big], \quad i = \overline{2,p}$$

and

$$\mu(A_1) \geq 2\mu(A_i) + \mu(X_i) - \mu(X_1), \quad i = \overline{2,p}.$$

Both the forms of (20) make it possible to perceive the process of reduction in $C(A)$ more deeply.

Based on Property 3 and Theorem 1 we shall derive some special results applied subsequently to the algorithms presented in Section 5. The following two propositions work under the assumptions of Theorem 1 and refer to the case where 'at most two subsets of X are non-empty' (this also includes the case $p = 2$).

Let A be a partition of T and the redistribution of tasks be performed between two groups (say, A_1 and A_2).

Proposition 1. *The necessary and sufficient condition for $C(A)$ to be reduced is*

$$\mu(A_1) > \mu(A_2) + \mu(X_2). \tag{24}$$

Proposition 2. *If (24) is satisfied, then the values of both $C(A)$ and $C_{\max}(A)$ will be reduced.*

Mathematical formulation $M2$ (12)–(15) is not the only possible. We present below the one that is not related to the algorithms discussed in this paper, but it can be useful in the development of scheduling algorithms. Let $x_1, \ldots, x_n$ be variables taking discrete values from 1 to p: $x_i \in \{1, \ldots, p\}$, $i = \overline{1,n}$. Introduce a function $f(x_1, \ldots, x_n)$ that is related to $C(A)$ according to the formula

$$f(x_1, \ldots, x_n) = C(A), \tag{25}$$

where $T_i \in A_l$ as $x_i = l$. Then the scheduling problem may be formulated as a combinatorial optimization problem

$$\min_{\substack{x_i \in \{1,\ldots,p\} \\ i=\overline{1,n}}} f(x_1, \ldots, x_n). \tag{26}$$

5. Algorithms that Minimize both $C(A)$ and $C_{\max}(A)$

Three propositions are presented below without proofs that are sufficiently simple and related to the results of Theorem 1 and Propositions 1 and 2. The results of these propositions are helpful in constructing simple minimization algorithms. Moreover, Propositions 3–5 determine necessary and sufficient conditions for the largest processing time to be reduced in the case of two groups of tasks.

Proposition 3. *Given a partition of tasks* $T_1, \ldots, T_r$ *into groups* $A_1, \ldots, A_p$, *transferring a task* $T_s \in A_k$ *to* A_l $(l \neq k)$ *implies a reduction in* $C(A)$ *if*

$$\mu(A_k) - \mu_s > \mu(A_l).$$

Proposition 4. *Suppose that tasks* $T_1, \ldots, T_r$ *are partitioned into groups* $A_1, \ldots, A_p$ *and analyze two subgroups* A_k^* *and* A_l^* *of tasks from* A_k *and* A_l $(l \neq k,\ A_k^* \subset A_k,\ A_l^* \subset A_l)$. *Interchanging* A_k^* *and* A_l^* *between* A_k *and* A_l *involves a reduction in* $C(A)$ *if*

$$\mu(A_k) - \mu(A_l) > \mu(A_k^*) - \mu(A_l^*). \tag{27}$$

Formula (27) determines the necessary properties of two groups of tasks A_k and A_l and two subgroups of tasks $A_k^* \subset A_k$ and $A_l^* \subset A_l$ for the reduction in $C(A)$ via interchanging the subgroups between the groups.

The following assertion follows as a partial case of Proposition 4.

Proposition 5. *Given a partition of tasks* $T_1, \ldots, T_r$ *into groups* $A_1, \ldots, A_p$, *analyze two tasks* $T_s \in A_k$ *and* $T_j \in A_l$ $(l \neq k)$. *Interchanging* T_s *and* T_j *between* A_k *and* A_l *implies a reduction in* $C(A)$ *if*

$$\mu(A_k) - \mu(A_l) > \mu_s - \mu_j > 0.$$

Two algorithms for minimizing $C(A)$ are presented and investigated below. They are based on the following strategies:

- an analysis of tasks in consecutive order and a search for a group where to transfer a separate task so as to decrease the value of $C(A)$ (Algorithm $P1$),
- an analysis of pairs of tasks from different groups for their further interchanging (Algorithm $P2$).

These algorithms are a realization of the well-known descent search: 'go as long as the goal function decreases.' They use special strategies in determining when a task must be transferred from one group to another or an interchange of tasks between two groups must be performed. The algorithms terminate when the actions by the selected strategy no longer decrease the value of $C(A)$.

Algorithm $P1$*:*

- At first, all the groups $A_1, \ldots, A_p$ are filled out using an algorithm of the initial partitioning of tasks.
- Analyze the tasks in consecutive order and search for a group A_l of transferring an individual task T_s (let $T_s \in A_k$, $l \neq k$) in order to reduce the value of $C(A)$: the transfer starts if

$$\mu(A_k) - \mu_s > \min_{\substack{1 \leq l \leq p \\ l \neq k}} \mu(A_l).$$

- The algorithm stops when the transfer of any task no longer decreases the value of $C(A)$.

Algorithm P2:

- At first, all the groups $A_1, \ldots, A_p$ are filled out using an algorithm of the initial partitioning of tasks.
- Analyze the pairs of tasks ($T_s \in A_k$, $T_j \in A_l$, $l \neq k$, $s < j$) in consecutive order and seek a pair of groups A_k and A_l for a possible interchange of tasks T_s and T_j so as to reduce the value of $C(A)$: the interchange starts if

$$\mu(A_k) - \mu(A_l) > \mu_s - \mu_j > 0.$$

- The algorithm stops when the transfer of any task no longer decreases the value of $C(A)$.

Algorithms $P1$ and $P2$ require an initial partition of tasks. Let us denote by $U1$ the following algorithm of initial partitioning:

- Initially, all the groups $A_1, \ldots, A_p$ are empty.
- Consider tasks T_i, $i = \overline{1,r}$ in consecutive order and put the current task T_c into a group with the smallest number of tasks.

Algorithm $U1$ is very simple and yields a schedule that is far from being optimal. Graham (1969) determined the best possible bounds for $U1$: algorithm $U1$ produces a schedule that has a makespan at most $(2-1/p)C^*_{\max}(A)$. If Graham's algorithm G is used to get the initial partition of tasks for a further analysis employing the algorithms based on Propositions 4–5, then the result of G may be improved.

Taking into account the results of the previous section we can draw the following conclusion: The algorithms which minimize $C(A)$ by using the results of Propositions 3–5 also tend to minimize the maximum finishing time $C_{\max}(A)$, because at any step, for an arbitrarily selected pair of groups (A_i, A_j), $i \neq j$, they try to minimize the maximum finishing time of tasks in these two groups, $\max\{\mu(A_i), \mu(A_j)\}$. Different schedules yielding different values of $C(A)$ may yield the same value of $C_{\max}(A)$.

6. Experimental Comparison of Algorithms

Four algorithms have been investigated: G, $U1{+}P1$, $G{+}P1$ and $U1{+}(P1{+}P2)$. The second and third algorithms use $U1$ and G, respectively, for the initial partitioning of tasks; further optimization is performed by $P1$. $(P1 + P2)$ means a combination of $P1$ and $P2$: $P1 + P2 + P1 + \ldots$ as long as any transfer of a task from one group to another or any interchange of two tasks from different groups do not reduce the value of $C(A)$ (and the value of $C_{\max}(A)$, cf. Property 3).

The experiments were carried out on sets of 100 randomly generated problems for different numbers r of tasks (up to $r = 1500$), and the results were averaged. The quantities $\mu_i \in [1, 100]$, $i = \overline{1,r}$ were generated at random, i.e. the situation was explored where the tasks with various execution times appeared with the same probability in the interval $[1, 100]$.

Figure 2 illustrates the difference between the values of $C_{\max}(A)$ obtained by G, $U1+P1$ and $U1+(P1+P2)$, in case $= 3$. Grey rectangles correspond to the difference $C_{\max}[U1 + (P1 + P2)] - C_{\max}[G]$, and white rectangles correspond to the difference $C_{\max}[U1 + P1] - C_{\max}[G]$, where $C_{\max}[S]$ denotes the value of $C_{\max}(A)$ obtained by algorithm S. Taking into account that the average values of $C_{\max}[U1 + (P1 + P2)]$, $C_{\max}[G]$ and $C_{\max}[U1 + P1]$ obtained during the experiments are equal to 6751.76, 6752.26 and 6752.56, respectively, we may conclude that the partitioning quality of these three algorithms is approximately the same. Attempts to make more complex tests by setting the values μ_i of five randomly selected tasks to be equal to 250 and generating other values of μ_i at random in [1,,100] led to similar results as in Fig. 2.

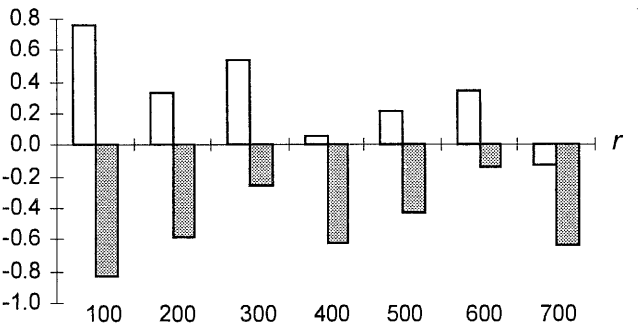

Fig. 2. The difference between the values of $C_{\max}(A)$ obtained by G, $U1 + P1$ and $U1 + (P1 + P2)$.

Figure 3 shows dependences of the average computing time t (in seconds on a 100 MHz PC) on the number r of tasks to be scheduled and on the number p of processing units for G and $U1 + P1$, respectively. This implies that the speed of Algorithm $U1 + P1$ depends essentially on p. The speed of G weakly depends on p because the main burden in G is related to the arrangement of tasks in decreasing order of their execution time. In our experiments, the quick-sort algorithm was used. Such sorting is independent of p.

The experiments showed that:

- Algorithm G yields a partition of tasks that cannot be improved by $P1$, but can be improved by $P2$. This means that G delivers a schedule that cannot be improved by transferring a single task from one group to another, but can be improved by interchanging tasks between two groups.

- The partitioning quality of Algorithms G, $U1 + P1$ and $U1 + (P1 + P2)$ is

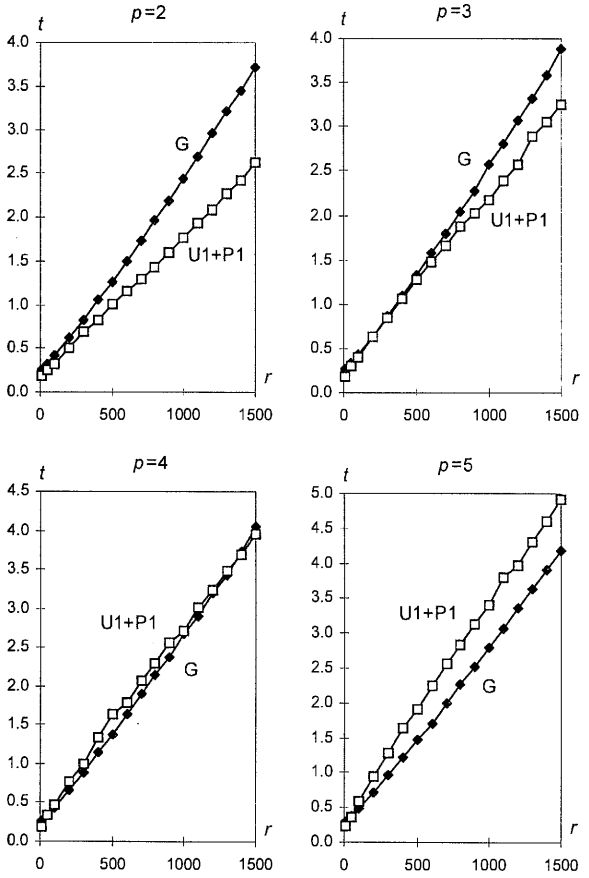

Fig. 3. Dependence of the average computing time t (in second) on the number r of tasks and on the number p of processing units.

similar; G yields a slightly better partition than that of $U1{+}P1$; $U1{+}(P1{+}P2)$ yields a slightly better partition than that of G.

- Algorithm $U1 + P1$ operates significantly faster than $U1 + (P1 + P2)$ (about 15–200 times depending on the number of tasks) because $P2$ is very time-consuming.

- Algorithm $U1 + P1$ operates faster than G in the case of a small number p of processing units ($p = 2, 3$, and, in some cases, $p = 4$ and even $p = 5$). The advantages of $U1 + P1$ over G are observed mostly for a larger number of tasks.

7. Conclusions

The results of this paper extend the set of criteria which characterize the scheduling quality as well as the set of possible scheduling strategies. An interaction between the processes of minimizing the new criteria and the maximum finishing time (the makespan of the schedule) has been discovered. We showed a possibility of minimizing the maximum finishing time by minimizing the new criteria that characterize the mean squared load of processing units. Moreover, the optimal workload of processing units found with the use of the proposed criteria is usually smoother (more balanced) than that found for traditional ones.

Some possible approaches to the minimization of the proposed criteria have also been discussed. The algorithms investigated are a realization of the well-known and sufficiently simple descent search. Therefore, their efficiency is similar to that of the classical algorithms. One of the reasons of such a choice is that a solution of the scheduling problem is usually a part of some more general problem. Sometimes this general problem requires much more computing time as compared with the solution to the scheduling problem, and the scheduling accuracy does not play the essential role. This is a domain of application of simple and easily realizable algorithms.

Clearly, the proposed algorithms are not the only possible ones to minimize the new criteria. More sophisticated strategies may also be acceptable, e.g. application of approximate methods that use task insertions and task swaps (Hubsher and Glover, 1994) or formulation of the problem as a combinatorial optimization one of type (25)–(26) and its solution by using simulated annealing (Dzemyda, 1996). In this paper, we apply only some possibilities from those provided by Theorem 1. Further investigations could lead to developing new, more effective algorithms.

Acknowledgement

The author wishes to thank the anonymous referees for many detailed and helpful comments, as well as for stimulating ideas for further investigations. The author would like to express his particular gratitude to Prof. Czesław Smutnicki (Wrocław University of Technology, Institute of Engineering Cybernetics) for the discussion regarding the final version of the paper.

References

Blazewicz J., Dror M. and Weglarz J. (1991): *Mathematical programming formulations for machine scheduling—A survey.* — Europ. J. Op. Res., Vol.51, No.3, pp.283–300.

Brucker P. (1998): *Scheduling Algorithms.* — Berlin/Heidelberg: Springer.

Dzemyda G. (1996): *Clustering of parameters on the basis of correlations via simulated annealing.* — Control and Cybernetics, Special Issue on *Simulated Annealing Applied to Combinatorial Optimization*, Vol.25, No.1, pp.55–74.

Graham R.L. (1969): *Bounds on the multiprocessing timing anomalies.* — SIAM J. Appl. Math., Vol.17, No.2, pp.416–429.

Hochbaum D.S. and Shmoys D.B. (1987): *Using dual approximation algorithms for scheduling problems: Theoretical and practical results.* — J. Assoc. Comp. Mach., Vol.34, No.1, pp.144–162.

Hochbaum D.S. and Shamir R. (1990): *Minimizing the number of tardy job units under release time constraints.* — Discr. Appl. Math., Vol.28, No.1, pp.45–57.

Hochbaum D.S. and Landy D. (1997): *Scheduling with batching: Two job types.* — Discr. Appl. Math., Vol.72, Nos.1–2, pp.99–114.

Hubsher R. and Glover F. (1994): *Applying tabu search with influential diversification to multiprocessor scheduling.* — Comp. Op. Res., Vol.21, No.8, pp.877–884.

Lam M. (1988): *Software pipelining: An effective scheduling techniques for VLIW machines.* — Proc. SIGPLAN'88 Conf. *Programming Language Design and Implementation*, Atlanta, Georgia, USA, pp.318–328.

Lilja D.J. (1991): *Architectural Alternatives for Exploiting Parallelism.* — Los Alamitos, CA: IEEE Computer Society Press.

Nawrocki J.R., Czajka A. and Complak W. (1998): *Scheduling cyclic tasks with binary periods.* — Inf. Process. Lett., Vol.65, No.4, pp.173–178.

Received: 16 December 1998
Revised: 12 June 1999
Re-revised: 27 September 1999
Re-re-revised: 11 October 1999