Scheduling Multiprocessor Tasks to Minimize Schedule Length

JACEK BŁAŻEWICZ, MIECZYSŁAW DRABOWSKI, AND JAN WĘGLARZ

Abstract — **The problem considered in this paper is the deterministic scheduling of tasks on a set of identical processors. However, the model presented differs from the classical one by the requirement that certain tasks need more than one processor at a time for their processing. This assumption is especially justified in some microprocessor applications and its impact on the complexity of minimizing schedule length is studied. First we concentrate on the problem of nonpreemptive scheduling. In this case, polynomial-time algorithms exist only for unit processing times. We present two such algorithms of complexity $O(n)$ for scheduling tasks requiring an arbitrary number of processors between 1 and k at a time where k is a fixed integer. The case for which k is not fixed is shown to be NP-complete. Next, the problem of preemptive scheduling of tasks of arbitrary length is studied. First an algorithm for scheduling tasks requiring one or k processors is presented. Its complexity depends linearly on the number of tasks. Then, the possibility of a linear programming formulation for the general case is analyzed.**

Index Terms — **Complexity analysis, deterministic scheduling, linear programming approach, microprocessor systems, polynomial-in-time algorithms, preemptive and nonpreemptive schedules, schedule length criterion, scheduling multiprocessor tasks.**

I. INTRODUCTION

ONE of the assumptions commonly imposed in machine scheduling theory is that each task is processed on at most one machine at a time. In fact, all polynomial-time algorithms as well as NP-completeness results for scheduling on machines (processors) were obtained on this assumption (see [3], [7], [12], [14], and [16] as surveys).

However, in recent years, with the rapid development of microprocessor and especially multimicroprocessor systems, the above assumption has ceased to be justified in some important applications. There are, for example, self-testing multimicroprocessor systems in which one processor is used to test others or diagnostic systems in which testing signals stimulate the tested elements and their corresponding outputs are simultaneously analyzed [2], [8]. When formulating scheduling problems in such systems, one must take into account the fact that some tasks have to be processed on more than one processor at a time. These problems create a new direction in machine scheduling theory. Preliminary results concerning the preemptive scheduling of tasks requiring one or two processors were obtained in [4].

Manuscript received June 4, 1984; revised December 10, 1984 and June 11, 1985.

J. Błażewicz and J. Węglarz are with Instytut Automatyki, Politechnika Poznańska, Poznań, Poland.

M. Drabowski is with Ośrodek Badawczy Zjednoczenia MERA, Kraków, Poland.

IEEE Log Number 8407304.

In this paper we study the computational complexity of nonpreemptive and preemptive scheduling to minimize schedule length for the general case of tasks requiring k processors at a time. Before doing this, we will set up the subject more precisely.

In general, we will consider a set of n *tasks* to be processed on a set of m *identical processors* $P = \{P_1, P_2, \cdots, P_m\}$. The set of tasks is divided into k (or fewer) subsets $T^1 = \{T_1^1, T_2^1, \cdots, T_{n_1}^1\}$, $T^2 = \{T_1^2, T_2^2, \cdots, T_{n_2}^2\}, \cdots, T^k = \{T_1^k, T_2^k, \cdots, T_{n_k}^k\}$ where $n = n_1 + n_2 + \cdots + n_k$. Each task T_i^1, $i = 1, 2, \cdots, n_1$, requires one arbitrary processor for its processing and its processing time is equal to t_i^1. On the other hand, each task T_i^j requires j arbitrary processors simultaneously for its processing during a period of time whose length is equal to t_i^j. Thus, we will call tasks from T^j *width-j tasks* or *T^j-tasks*. Scheduling of tasks is termed *preemptive* scheduling if processing of any task from the set $T^1 \cup T^2 \cup \cdots \cup T^k$ can be arbitrarily preempted and restarted later at no cost, and *nonpreemptive* scheduling if this is impossible. All the tasks considered in the paper are assumed to be *independent*, i.e., there are no precedence constraints among them. A schedule will be called *feasible* if, besides the usual conditions, each T^1 task is processed by one processor and each T^k task is processed by k processors at a time. A feasible schedule will be *optimal* if it is of minimum length. A scheduling algorithm is an *optimization* one if it always finds an optimal schedule.

In Section II we will analyze the complexity of nonpreemptive scheduling of T^1 tasks and $T^2, \cdots, T^k$ tasks for the case of unit processing times. (The case of arbitrary processing times is NP-complete, because the scheduling of T^1 tasks with arbitrary processing times is known to be NP-complete [7].[1]) In Section III we will present a linear time algorithm for solving the problem of preemptive scheduling of T^1 and T^k tasks. Then, a linear programming formulation for the scheduling of $T^1, T^2, \cdots$ and T^k tasks is analyzed.

II. NONPREEMPTIVE SCHEDULING

In this Section we will analyze the complexity of scheduling in a nonpreemptive mode. As we mentioned earlier, we will focus our attention on unit processing time tasks. Let us observe that the latter problem is equivalent to the bin packing problem with the restrictions on the magnitudes of the numbers to be packed. Thus, several known approximation algorithms for the bin packing problem (see [11] for a survey)

[1]We assume that the reader is familiar with the general concepts of the NP-completeness theory which may be found, for example, in [10].

0018-9340/86/0500-0389$01.00

can be used in this case. However, it is also possible to devise new algorithms for some cases not considered yet. Let us start with the problem of scheduling tasks which belong to two sets only: T^1 and T^k, for arbitrary k. This problem can be solved optimally by the following algorithm.

Algorithm 1: 1) Calculate the length of an optimal schedule according to the formula[2]

$$C^*_{\max} = \max\left\{\left\lceil \frac{n_1 + kn_k}{m} \right\rceil, \left\lceil \frac{n_k}{\left\lfloor \frac{m}{k} \right\rfloor} \right\rceil\right\} \tag{1}$$

2) Schedule the width-k tasks first in time interval $[O, C^*_{\max}]$. Then assign unit-width tasks to the remaining free processors.

It should be clear that (1) gives a lower bound on the schedule length of an optimal schedule and this bound is always met by a schedule constructed by Algorithm 1. The two possible forms of this schedule, depending upon which of the two quantities in (1) is maximum, are shown in Fig. 1. It follows that Algorithm 1 optimizes schedule length and its complexity is $O(n)$.

Let us consider now the case of scheduling tasks belonging to sets $T^1, T^2, T^3, \cdots, T^k$ where k is a fixed integer. The approach used for solving the problem is similar to that for solving the problem of nonpreemptive scheduling of unit processing time tasks under fixed resource constraints [5]. We describe its modification below.

First, let us observe that each instance I of our scheduling problem can also be presented as an input vector $\mathbf{t}_I = [b_{I1}, b_{I2}, \cdots, b_{In}]$, whose component b_{Ii} gives the number n_i of T^i tasks. By a *processor feasible set* we mean a set of tasks $T^1, T^2, T^3, \cdots, T^k$, for which the following inequality is met.

$$n_1 + 2n_2 + 3n_3 + \cdots + kn_k \leqslant m.$$

It is obvious that tasks forming an arbitrary processor feasible set can be processed in one unit of time. Moreover, since k is fixed, it follows that the number K of different processor feasible sets is also fixed. The input vectors which correspond to these sets (we will call them elementary vectors) are denoted by $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_K$.

Now, the problem is to find a decomposition of a given instance into the minimum number of processor feasible sets. It is clear that this is equivalent to finding a decomposition of vector $\mathbf{n} = (n_1, n_2, \cdots, n_k)$ into a linear combination of elementary vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_K$. Thus, we have the following problem:

Minimize

$$\sum_{j=1}^{K} x_j$$

subject to

$$\sum_{j=1}^{K} x_j \mathbf{b}_j = \mathbf{n} \quad x_j \geqslant 0 \quad \text{and} \quad x_j \text{ integer.}$$

[2] $\lfloor x \rfloor$ denotes the greatest integer not greater than x, and $\lceil x \rceil$ denotes the smallest integer not smaller than x.

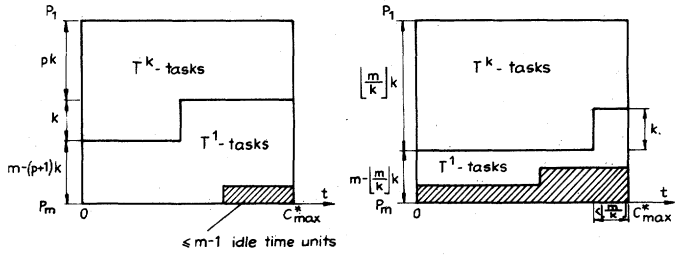

Fig. 1. Two possible forms of an optimal schedule.

Hence, we obtain an integer linear programming formulation for the problem with a fixed number of variables. From [15] we know that the integer linear programming problem with a fixed number of variables K can be solved in polynomial time in the number of constraints M and $\log a$ where a is the largest coefficient in the linear integer programming problem. The complexity of this problem is $O(2^{K^2}(KM \log a)^{c^K})$, for some constant c. In our case, we obtain the final complexity $O((\log n)^{c^K}) < O(n)$.

It is also possible to solve the above problem via a dynamic programming approach. Since in this case there are no more that $O(m^k)$ different processor feasible sets a straightforward way leads to a dynamic programming optimization algorithm whose complexity is $O(m^{k-1}n^k)$.

Now, let us consider the problem of scheduling tasks from sets $T^1, T^2, T^3, \cdots, T^k$ where k is an arbitrary (and not fixed) integer. Unfortunately, this problem (its decision version) is NP-complete in the strong sense, and this fact follows the NP-completeness of the 3-partition [9].

From the above and the remark concerning the complexity of nonpreemptive scheduling of tasks of arbitrary length, we see that many scheduling problems involving nonpreemptable tasks are NP-complete and therefore intractable in practice. For this reason in the next section we consider the problem of preemptive scheduling of multiprocessor tasks.

III. Preemptive Scheduling

We will start this Section by analyzing the problem of preemptive scheduling of arbitrary processing time tasks from sets T^1 and T^k in order to minimize the schedule length. Let us first prove a lemma that specifies the form of this schedule. That is, we will prove that among minimum-length schedules there exists a feasible *A-schedule,* i.e., one in which first all T^k-tasks are assigned in time interval $[O, C^*_{\max}]$ using McNaughton's rule, and then all T^1-tasks are assigned using the same rule, in the remaining part of the schedule (see Fig. 2).

Lemma 1: For any feasible schedule, there exists a corresponding feasible A-schedule of the same length.

Proof: Let us suppose that some feasible schedule of length equal to $C_{\max}$ is not an A-schedule. Three types of operations can be specified for each part of task T^k_i, $i = 1, 2, \cdots, n_k$, processed in that feasible schedule (see Fig. 3).

1) The operation of the first type consists of joining all the parts of any task that are processed in time interval τ_i to process them on consecutive processors.

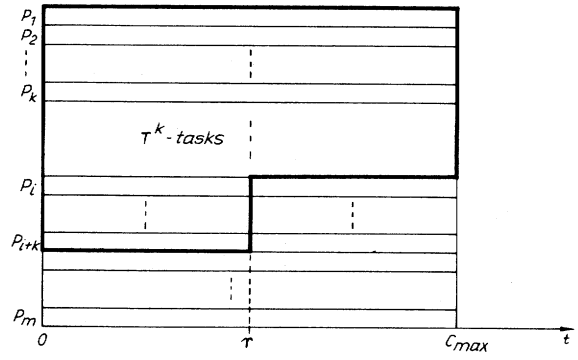

Fig. 2. An A-schedule.

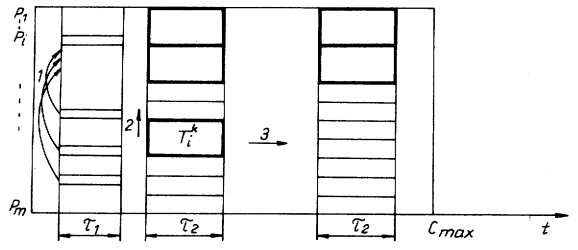

Fig. 3. An arbitrary feasible schedule and three types of operations that can be defined for a part of task T_i^k.

2) The operation of the second type consists of moving the part of task T_i^k up in the schedule and in exchanging its position with T^1-tasks (or their parts).

3) The operation of the third type consists of moving the part of T_i^k forward or backward in the schedule to the place (period of time) where more T^1-tasks are processed than in the previous period, and in exchanging its position with k T^1-tasks (or idle time tasks).

It is rather clear that all these operations are feasible and none of them will lengthen the schedule. Thus, it is also not hard to see that after a finite number of these operations we will get an A-schedule which is feasible and whose length is equal to C_{max}. □

Following the above lemma we will concentrate on finding an optimal schedule among A-schedules. Now, we give a lower bound on schedule length for our problem. Define

$$X = \sum_{i=1}^{n_1} t_i^1, \quad Y = \sum_{i=1}^{n_k} t_i^k, \quad Z = X + kY$$

$$t_{\max}^1 = \max\{t_i^1 : T_i^1 \in T^1\}, \quad t_{\max}^k = \max\{t_i^k : T_i^k \in T^k\}.$$

A bound on C_{max} is obtained as follows:

$$C_{max} \geq C = \max\{Z/m, Y/\lfloor m/k \rfloor, t_{\max}^1, t_{\max}^k\}. \quad (2)$$

It is clear that no feasible schedule can be shorter than the maximum of the above factors, i.e., mean-processing requirement on one processor, mean-processing requirement of T^k-tasks on k processors, maximum processing time of a T^1-task and maximum processing time of a T^k-task. If $m \cdot C > Z$, there will be an idle time in any schedule and its minimum length $IT = m \cdot C - Z$.

On the basis of bound (2) and Lemma 1 one can try to construct a preemptive schedule of minimum length equal to C. However, it is not always possible, as illustrated by the following example. Let $n = 11$, $m = 23$, $k = 7$, $n_1 = 8$, $n_7 = 3$, $\mathbf{t}^1 = [6, 6, 6, 6, 5, 5, 5, 1]$, $\mathbf{t}^7 = [6, 6, 2]$. Using the above formula we obtain $X = 40$, $Y = 14$, $Z = 138$, the lower bound on schedule length $C = \max\{138/23, 14/3, 6, 6\} = 6$ and the length of idle time $IT = 0$. A partial schedule is shown in Fig. 4. From this figure we can see that it is not possible to construct a feasible schedule of the length equal to 6. This can be seen from the fact that there are four width-1 tasks of length 6, each of which must be continuously processed in the interval $[0, 6]$. Yet no more than two of these width-1 tasks can be processed simultaneously in the interval $[0, 2]$. In general, the schedule must be long enough so that the amount of processing that must be done on any subset of tasks in the interval $[0, r]$ does not exceed the total amount of processing that can be done in that interval. Below we will present the reasoning that allows one to find the optimal schedule length.

Let $p = \lfloor Y/C \rfloor$. It is quite clear that the optimal schedule length $C_{\max}^*$ must obey the following inequality:

$$C \leq C_{\max}^* \leq \frac{Y}{P}.$$

By Lemma 1 there exists an optimal A-schedule with $k \cdot p$ processors devoted entirely to T^k-tasks, k processors devoted to T^k-tasks in time interval $[0, r]$ and T^1 tasks scheduled in the remaining time (see Fig. 2). Let the number of processors that can process T^1-tasks in time interval $[0, r]$ be

$$m_1 = m - (p + 1) \cdot k.$$

An A-schedule which completes all tasks by time D where $C \leq D \leq Y/P$, will have $r = Y - Dp$. Thus, the optimum value $C_{\max}^*$ will be the smallest value of D ($D \geq C$) such that the T^1-tasks can be scheduled on $m_1 + k$ processors available in the interval $[Y - Dp, D]$ and m_1 processors available in the interval $[0, D]$. Below we give necessary and sufficient condition for the unit-width tasks to be scheduled. To do this, let us assume that these tasks are ordered in such a way that $t_1^1 \geq t_2^1 \geq \cdots \geq t_{n_1}^1$. For a given pair D, r ($r = Y - Dp$), let $t_1^1, t_2^1, \cdots, t_j^1$ be the only processing times greater than $D - r$. Then the T^1-tasks can be scheduled if and only if

$$\sum_{i=1}^{j} [t_i^1 - (D - r)] \leq m_1 r. \quad (3)$$

To prove that the above condition really is necessary and sufficient, let us first observe that if (3) is violated the T^1-tasks cannot be scheduled. Suppose now that (3) holds, then one should schedule the excess (exceeding $D - r$) of long tasks $T_1^1, T_2^1, \cdots, T_j^1$ and (if (3) holds without equality) some other tasks on m_1 processors in time interval $[0, r]$ using McNaughton's rule [17]. After this operation the interval is completely filled with unit-width tasks on m_1 processors.

Now, we describe how the optimum value of the schedule length ($C_{\max}^*$) can be found. Let $P_j = \sum_{i=1}^{j} t_i^1$. Inequality (3) may be rewritten as

$$P_j - j(D - |Y - Dp|) \leq m_1(Y - Dp).$$

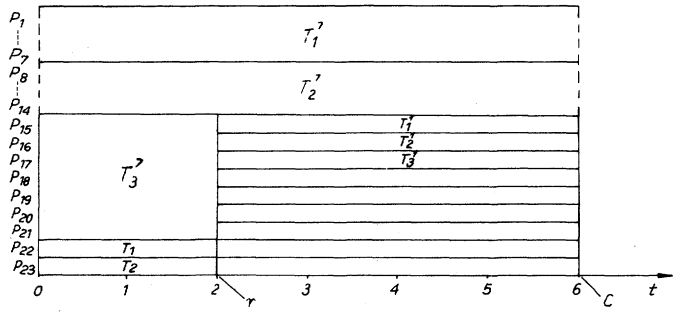

Fig. 4. A preliminary schedule for Example 1.

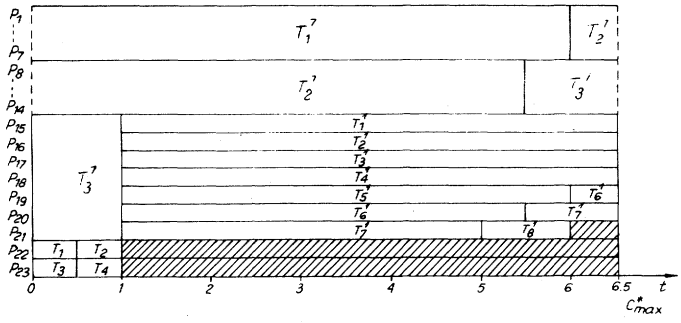

Fig. 5. Example 1 continued: An optimal schedule.

Solving it for D we get

$$D \geqslant \frac{(j - m_1)Y + P_j}{(j - m_1)p + j}.$$

Define

$$C_j = \frac{(j - m_1)Y + P_j}{(j - m_1)p + j}.$$

Thus, we may write:

$$C^*_{\max} = \max\{C, C_1, C_2, \cdots, C_{n_1}\}.$$

Finding the above maximum can clearly be done in $O(n_1 \log n_1)$ time by sorting the unit-width tasks by t_i. But one can do better by taking into account the following facts.

1) $C_i \leqslant C$ for $i \leqslant m_1$ and $i \geqslant m_1 + k$.

2) C_i has no local maxima for $i = m_1 + 1, \cdots, m_1 + k - 1$.

Thus, to find a maximum over $C_{m_1=k-1}, \cdots, C_{m_1=k-1}$ and C we only need to apply a linear time median finding algorithm [1] and binary search. This will result in an $O(n_1)$ algorithm that calculates $C^*_{\max}$. This is because finding the medians takes $O(n_1)$ the first time, $O(n_1/2)$ the second time, $O(n_1/4)$ the third time, $\cdots$. Thus, the total time to find medians is $O(n_1)$.

Now we can give an optimization algorithm for the considered case of scheduling unit-width and width-k tasks.

Algorithm 2:

1) Calculate the minimum schedule length $C^*_{\max}$.

2) Schedule all the T^k tasks (in any order) in interval $[O, C^*_{\max}]$ using McNaughton's rule.

3) Assign the excess of the long tasks (that exceed $C^*_{\max} - r$) and possibly some other tasks to the m_1 processors in the interval $[0, r]$. Schedule the remaining processing requirement in the interval $[r, C^*_{\max}]$ using McNaughton rule. □

The optimality of the above algorithm follows from the discussion preceding it. Its complexity is $O(n_1 + n_k)$, thus we get $O(n)$. Let us consider our example once more. Using the above algorithm, we get the optimal schedule given in Fig. 5.

Let us turn our attention to the problem of preemptive scheduling of tasks from sets $T^1, T^2, T^3, \cdots, T^k$. It may be formulated as a linear programming (LP) problem in the way similar to that of resource constrained scheduling [18]. Below, we describe briefly a modified LP formulation of the problem. By a *processor feasible set* we mean here a set of tasks which can be processed simultaneously. Let the number of different processor feasible sets be equal to M. By x_i we denote the processing time of the ith processor feasible set and by Q_j the set of indexes of those processor feasible sets which contain task J_j for

$$J_j \in \{T^1_1, \cdots, T^1_{n_1}, T^2_1, \cdots, T^2_{n_2}, \cdots, T^k_1, \cdots, T^k_{n_k}\}.$$

Thus, the following linear programming problem can be formulated: Minimize

$$\sum_{i=1}^{M} x_i$$

subject to

$$\sum_{i \in Q_j} x_i = t^k_j, \quad J_j \in \{T^r_1, \cdots, T^r_{n_r}\}, \quad r = 1, 2, \cdots, k.$$

As the solution of the above problem we get optimal values x_i^* of partial schedule lengths in an optimal schedule. The tasks processed in these partial schedules are members of the corresponding processor feasible subsets. Let us calculate the complexity of the above approach. The number of constraints is equal to n and the number of variables is $O(n^m)$. Thus, for a fixed number of processors it is bounded from above by a polynomial in the number of tasks. On the other hand, a linear programming problem may be solved using Khachijan's algorithm [13] in time bounded from above by a polynomial in the number of variables, the number of constraints, and the sum of logarithms of all the coefficients in the LP problem. Thus, our scheduling problem may be solved in polynomial time for a fixed number of processors.

IV. Conclusions

The model of scheduling k-processor tasks presented can be extended in many ways. Processors of different speeds as well as dedicated processors can be included. Precedence constraints among tasks can be introduced. The model can also be extended by introducing additional resources [5], [6], [9]. The above issues are now being studied.

Acknowledgment

The authors are indebted to the anonymous referees for their valuable comments, especially for improving the algorithms in Section III.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] A. Avizienis, "Fault tolerance: The survival attribute of digital systems," *Proc. IEEE*, vol. 66, no. 10, pp. 1109–1125, 1978.

[3] K. R. Baker, *Introduction to Sequencing and Scheduling*. New York: Wiley, 1974.

[4] J. Błażewicz, M. Drabowski, and J. Węglarz, "Scheduling independent 2-processor tasks to minimize schedule length," *Inform. Processing Lett.*, vol. 18, pp. 267–273, 1984.

[5] J. Błażewicz and K. Ecker, "A linear time algorithm for restricted bin packing and scheduling problems," *Oper. Res. Lett.*, vol. 2, no. 2, pp. 80–83, 1983.

[6] J. Błażewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Appl. Math.*, vol. 5, no. 1, pp. 11–24, 1983.

[7] E. G. Coffman, Jr., Ed., *Computer and Job/Shop Scheduling Theory*. New York: Wiley, 1976.

[8] M. Dal Cin and E. Dilger, "On the diagnosability of self-testing multiprocessor systems," *Microprocessing Microprogramming*, vol. 7, no. 3, pp. 177–184, 1981.

[9] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM J. Comput.*, vol. 4, pp. 397–411, 1975.

[10] ——, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[11] ——, "Approximation algorithms for bin packing problems: A survey," in *Analysis and Design of Algorithms in Combinatorial Optimization*, G. Ausiello and M. Lucertini, Eds. Vienna, Austria: Springer-Verlag, 1982, pp. 147–172.

[12] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling theory: A survey," *Ann. Discrete Math.*, vol. 5, pp. 287–326, 1979.

[13] L. G. Khachian, "A polynomial algorithm for linear programming" (in Russian), *Doklady Akad. Nauk USSR*, vol. 244, pp. 1093–1096, 1979.

[14] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Recent developments in deterministic sequencing and scheduling: A survey," in *Deterministic and Stochastic Scheduling*, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, Eds. Dordrecht, The Netherlands: Reidel, 1982, pp. 35–73.

[15] H. W. Lenstra, Jr., "Integer programming with a fixed number of variables," Univ. Amsterdam, The Netherlands, Tech. Rep., 1981.

[16] J. K. Lenstra and A. H. G. Rinnooy Kan, "Scheduling theory since 1981: An annotated bibliography," Mathematisch Centrum, Amsterdam, The Netherlands, Rep. BW 188/83, 1983.

[17] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Sci.*, vol. 12, no. 1, pp. 1–12, 1959.

[18] J. Węglarz, J. Błażewicz, W. Cellary, and R. Słowinski, "An automatic revised simplex method for constrained resource network scheduling," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 295–300, 1977.

Jacek Błażewicz was born in Poznań, Poland on August 11, 1951. He received the M.Sc., Ph.D., and Dr.habil. degrees, all in computer science, from the Technical University of Poznań, Poznań, Poland, in 1974, 1977, and 1980, respectively.

Since 1974 he has been with the Technical University of Poznań, where he is presently the Deputy Dean of the Electrical Engineering Faculty. His research interests are algorithm design and complexity analysis of algorithms especially in scheduling theory and in data transmission systems. He is the author and coauthor of a number of papers published in international journals and the coauthor of several books (in Polish) on the above subject.

Dr. Błażewicz is a member of the Council of the Polish Informatics Society, the Polish Cybernetical Society, and the American Mathematical Society.

Mieczysław Drabowski was born in Kraków, Poland, in 1953. He received the M.Sc. degree in control and communication engineering from the Academy of Mining and Metallurgy, Kraków, and the M.Sc. degree in mathematics from Jagiellonian University, Kraków, in 1977 and 1979, respectively.

He has eight years of industrial experience in software engineering, computer aided digital circuits testing, microdiagnostic computer systems, and floppy disk drivers and controllers design and testing. Currently, he is a Senior Development Engineer with the MERA-KFAP, Kraków, and manages a group responsible for the design of 16-bit microcomputer systems. His research interests include multiprocessor system architecture, fault-tolerant computing, and task scheduling.

Jan Węglarz was born in Poznań, Poland on September 24, 1947. He received the M.Sc., Ph.D., and Dr.habil. degrees from the Technical University of Poznań, Poznań, Poland.

He is now a Professor and Chairman of the Laboratory of Operations Research and Computer Systems, Technical University of Poznań. His research interests include widely understood scheduling and resource allocation problems in project, production, and computer systems. He has been an invited speaker to various universities and conferences, and has served as a reviewer to several publishers. He is also an Associate Editor of *Foundations of Control Engineering*, a member of the Editorial Board of the *European Journal of Operational Research*, and the *International Journal of Modeling and Simulation*.

Dr. Węglarz has received a number of awards for his scientific contributions and scholarship, among others, two awards from the Polish Academy of Sciences (PAS). He is a member of the Computer Science Committee of PAS, the Council of the Polish Informatics Society, the Polish Cybernetical Society, the Polish Mathematical Society, and the American Mathematical Society.