A strategy for scheduling splittable tasks to reduce schedule length

By J. BŁAŻEWICZ, W. CELLARY, J. WĘGLARZ

1. Introduction

In deterministic problems of scheduling tasks on processors (static job-shop problems) it is usually assumed that task execution times are known in advance. Of course, in practice this assumption is not always met, but even then the solution of deterministic problems of scheduling has an important practical meaning. Firstly, when the expected values of task execution times are known, then it is possible, using established techniques [3], to find an optimistic estimate of the expected value of the schedule length. Secondly, upper bounds of execution times of individual tasks may be known. Then scheduling using these values corresponds to the analysis of the worst case and is applied in hard-real-time problems with strict deadlines that must be observed.

Independently of this one can measure task execution times after processing a given set of tasks and use them to find an optimal schedule. This allows one to estimate an operational scheduler and to draw conclusions about possible improvments.

It becomes more and more important to schedule splittable (preemptable) tasks i.e. those that may be preempted; the processing of preempted task may resume where it left off without any extra work or time being occasioned by the preemption. Examining splittable tasks is of a great importance in systems of parallel processors using a common operating store. Such systems have increasingly many applications in the control of such processes as traffic, telephone switch control organization [5, 11] in which several processors using a common data base and computational procedures are being used. It is easy to verify that the possibility of preemption is profitable for improving the schedule length.

Scheduling splittable tasks was considered in [8, 9, 10]. Algorithms, presented in these papers, concern only homogeneous processors and relatively simple precedence relations among tasks. In [4, 7], the problem was considered of scheduling independent tasks on processors that are consistently fast or donsistently slow for all the tasks. In the papers mentioned above non-enumerative algorithms were presented. However the problem of scheduling dependent, splittable tasks, in the

general case, is known to be polynomial complete [4] and hence unlikely to admit a non-enumerative solution. Thus, for this case the direct use of scheduling strategies in an operating system has rather restricted applications. Finding such strategies has, however, practical significance for the following reasons. Firstly, one can use them to estimate an operational scheduler. Secondly, the distance between an optimal solution and a suboptimal one for a heuristic, non-enumerative approach, may be found. Lastly, enumerative algorithms may be used in computer centres that perform large and complex numerical computations but not in a real-time environment.

In this paper such a scheduling strategy will be presented, which gives some particular advantages. Then it will be compared for the case of homogeneous processors with the strategy described in [2].

2. Scheduling on heterogeneous processors

There are given a set of m processors $P_1, P_2, \ldots, P_m$ and a set of n tasks $T_1, T_2, \ldots, T_n$. The execution time of task T_j on processor P_i will be denoted by τ_{ij}, where τ_{ij} is a positive real number.

We will assume, that precedence relations among tasks are given in the form of an activity network in which arcs correspond to tasks and nodes to events. Let the number of nodes of the network be equal to $r+1$. It will be assumed that the events are ordered in such a way that event j does not occur earlier than event i if $i<j$.

The concept of the algorithm for scheduling splittable tasks on heterogeneous processors to minimize schedule length was given in [1]. For this purpose the following denotations were introduced:

— S_k, $k=1, 2, \ldots, r$, the set of all tasks which may be processed between the occurrence of event k and $k+1$. This set is called the main set;

— K_j, $j=1, 2, \ldots, n$, the set of indices of these main sets in which task T_j may be processed.

For a given schedule we denote:

— $x_{ijk} \in \langle 0, 1 \rangle$, $i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$; $k \in K_j$, a part of task T_j processed on processor P_i in S_k;

— $t_{ijk}=\tau_{ij} \cdot x_{ijk}$, the processing time of a part x_{ijk};

— $t_{ijk}=\sum_{i=1}^{m} t_{ijk}$, $j=1, 2, \ldots, n$; $k \in K_j$ the processing time of a part of task T_j processed in S_k;

— $t_j=\sum_{k \in K_j} t_{jk}$, $j=1, 2, \ldots, n$, the processing time of the whole task T_j;

— y_k, $k=1, 2, \ldots, r$, the schedule length in S_k;

— $y=\sum_{k=1}^{r} y_k$, the schedule length.

Using the above denotations, the following linear programming (LP) problem may be formulated:

Minimize y
Subject to

$$\sum_{k \in K_j} \sum_{i=1}^{m} x_{ijk} = 1 \quad j = 1, 2, \ldots, n, \tag{1}$$

$$y_k - \sum_{j \in S_k} x_{ijk} \tau_{ij} \geqq 0 \quad \begin{matrix} i = 1, 2, \ldots, m, \\ k = 1, 2, \ldots, r, \end{matrix} \tag{2}$$

$$y_k - \sum_{i=1}^{m} x_{ijk} \tau_{ij} \geqq 0 \quad \begin{matrix} j = 1, 2, \ldots, n, \\ k \in K_j. \end{matrix} \tag{3}$$

Equation (1) guarantees that every task will be processed; inequality (2) defines y_k's as the schedule lengths; inequality (3) assures that obtaining a feasible schedule will be a possible, i.e. one such that no task is processed simultaneously on more than one processor.

As the result of solving the described LP problem the optimal values y^*, x^*_{ijk}, t^*_{ijk} and t^*_j, $i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$; $k \in K_j$, are obtained. However, all starting points of parts of tasks are unknown. These points may be found by using the rule shown in Fig.1. As the initial values for the rule, the optimal values, obtained by solving the LP problem formulated above, are taken. In Fig. 1 $t(i)$, $i=1, 2, \ldots, m$, denotes the processing time t_{jk} of the i'th assigned task, and $t(m+1)$ the processing time t_{jk} of the first unassigned task.

3. Development of the algorithm

In the problem described in Section 2. the first feasible solution is known in advance — it is the sequential processing of all the tasks on a single processor.

The number of variables is $v=(m+1)\cdot(r+\sum_{j=1}^{n}|K_j|)$, where $|K_j|$ denotes the number of elements of set K_j. The number of constraints is $c=n+mr+\sum_{j=1}^{n}|K_j|$. For solving this problem the Revised Simplex Method [6] is worthwhile, because for most cases $v>3c$. It is clear that the number of variables and constraints increases with increasing of the number of tasks and processors. For example for 5 processors and not very complicated networks containing 10 tasks the number of variables is about 100, 30 tasks — 500, 60 tasks — $1.5\cdot10^3$ and 100 tasks — $5\cdot10^3$. The numbers of constraints for the same networks are respectively about 50, 200, 400 and 800. If we want to use directly one of the simplex methods for solving the LP problem, about 10^7 memory cells for 100 tasks will be needed because of the necessity of memorizing the matrix of coefficients which is the largest one in the problem. Thus the direct use of simplex methods has here a very restricted application.

Below an approach which allows for the great reduction of the difficulty mentioned above will be shown.

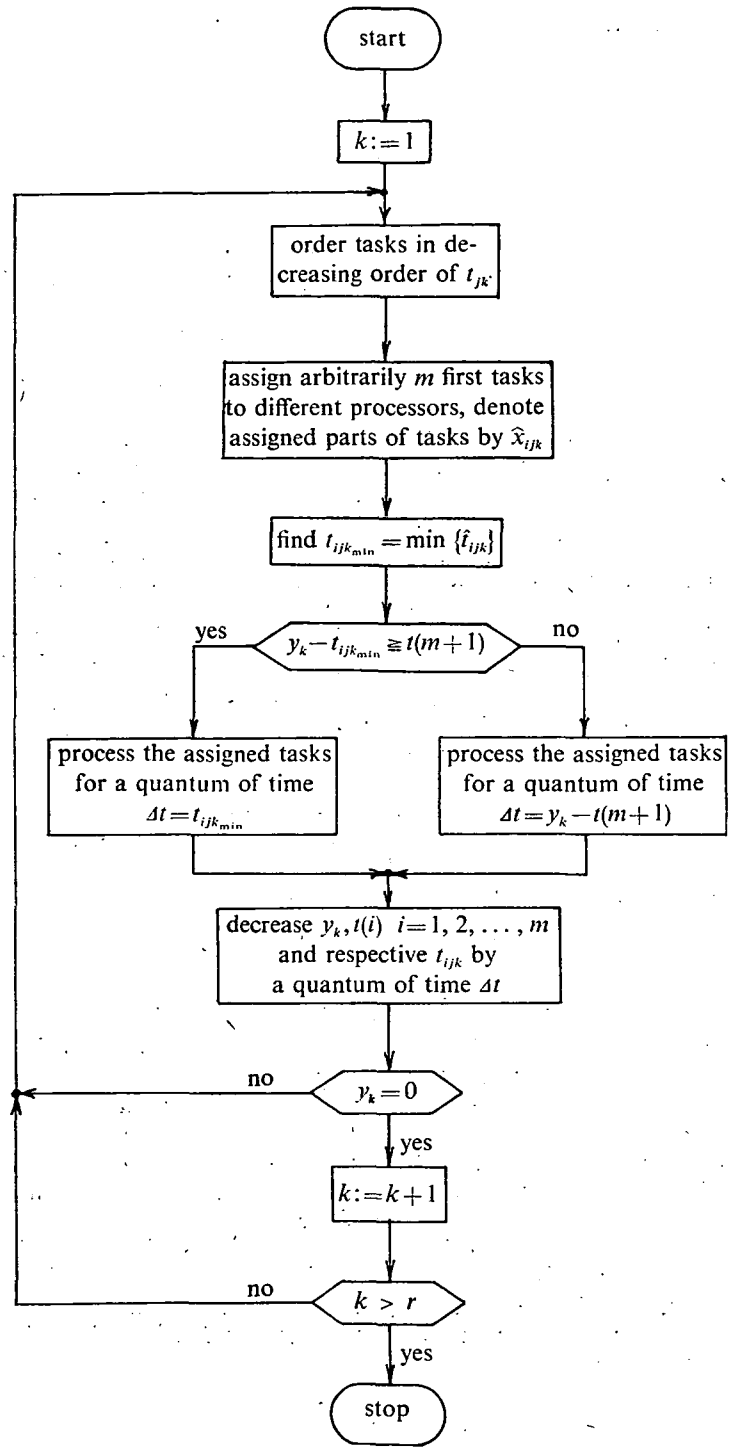

Fig. 1
Finding starting points of x^*_{ijk}'s

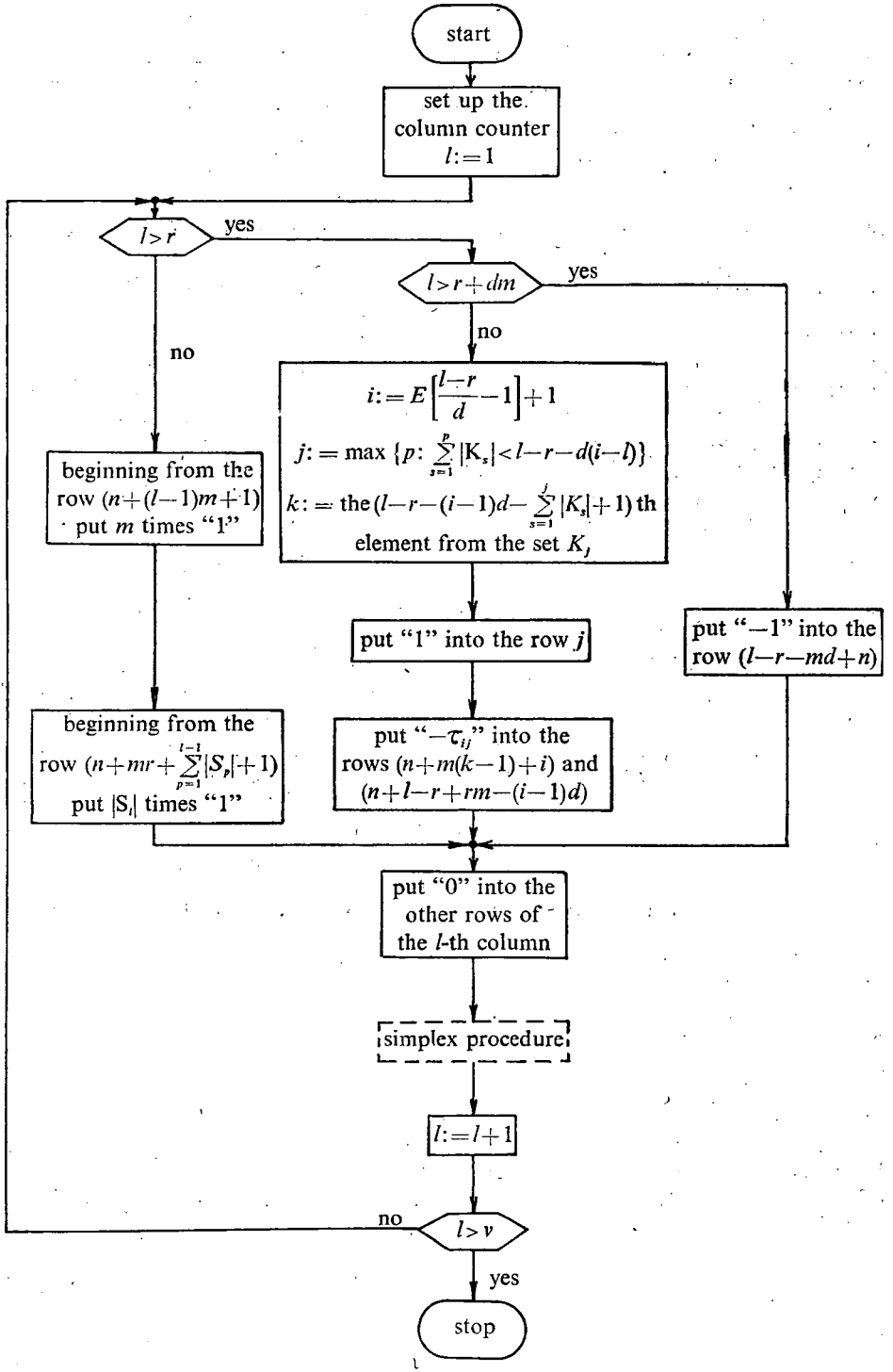

Fig. 2

Generation of consecutive columns of the matrix of coeffcients in one simplex iteration

The idea of this approach is based on a generation of consecutive columns of the matrix of coefficients in every simplex iteration. Such generation is possible, because in the Revised Simplex Method the elements of the matrix of coefficients are constant during computation. As a result only one column of this matrix has to be stored at any moment, and so storage requirements are significantly reduced.

For the purpose of describing the technical aspects of generation let us distinguish among columns of the matrix of coefficients three sets of columns. The first set contains r columns corresponding to variables y_k; the second set — $m \cdot \sum_{j=1}^{n} |K_j|$ columns corresponding to variables x_{ijk}; the third set — $mr + \sum_{j=1}^{n} |K_j|$ columns corresponding to artificial variables. After identification of the actually generated column to which the set does belong, appriopriate values are put into the rows corresponding to constraints (1), (2) and (3) on the base of the minimum information about the structure of the problem. This information includes n, m, matrix of execution times $[\tau_{ij}]$ and the vector describing the structure of the network, containing arcs as ordered pairs of nodes. After generation a single column, one check the benefit of introducing this column into the solution of the LP problem in accordance with the simplex procedure. The number of constraints (1), (2), (3) are equal respectively to n, $r \cdot m$ and $\sum_{j=1}^{n} |K_j|$. The block diagram of the generation of consecutive columns of the matrix of coefficients is one simplex iteration is shown in Fig. 2. In Fig. 2. $d = \sum_{j=1}^{n} |K_j|$.

The fact must be stressed that the computer time used by the algorithm in comparison with the time used by the algorithm in which the procedure of generation is not used, is reduced, except for small problems which do not require mass storage. Of course, if the network-node ordering is not given, the obtained schedule is in general a suboptimal one. The optimal schedule may be obtained by choosing the best one from among optimal solutions for all possible orders.

4. Scheduling on homogeneous processors — a comparison of two algorithms

In the case of homogeneous processors, tasks may be scheduled in accordance with the algorithm described in Sections 2 and 3. Let us call it the *A-algorithm*. However, for this case, a special algorithm has been elaborated [2] which will be called the *B-algorithm*. In this Section we present the conceptual basic of this algorithm in comparison with the *A*-algorithm.

In the *B*-algorithm we also use the concept of the main sets S_k, $k = 1, 2, \ldots, r$, which was introduced in Section 2.

Let us number from 1 to N the feasible sets, i.e. those subsets of all main sets, in which the number of elements is not greater than m. Now let Q_j denote the set of all numbers of the feasible sets in which task T_j may be processed and t_i the duration of set i. Thus one obtains the LP problem:

Minimize

$$y = \sum_{i=1}^{N} t_i.$$

Subject to

$$\sum_{i \in Q_j} t_i = \tau_j \quad j = 1, 2, \ldots, n \tag{4}$$

or in matrix notation

$$At = \tau$$

where A is the matrix of coefficients:

$$a_{ij} = \begin{cases} 1 & \text{if } i \in Q_j \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, the columns of matrix A correspond to the resource feasible sets. The number of variables in this problem is much greater than in A-algorithm, for example: for 5 processors and 10 tasks it is about 50, 30 tasks — $2 \cdot 10^3$, 60 tasks — $3 \cdot 10^4$ and 100 tasks — $2 \cdot 10^5$. On the other hand, the number of constraints is much smaller than in A-algorithm, because it is equal to the number of tasks (see (4)).

In order to avoid the storage of matrix A, the method of automatic generation of columns for B-algorithm, for this matrix was also elaborated [2].

Comparing these two algorithms one should pay attention to core store requirements and computer time.

Core store requirement for both algorithms is equal 16c. So in this respect, it is more worthwhile to use the B-algorithm, because the number of constraints c in it is much smaller. The number of variables as well as the number of constraints influence computer time. In Table 1 computer times for A- and B-algorithms are compared for not very complicated networks and 5 processors. These results were produced using programs written in FORTRAN IV and processed on an ODRA 1305.

Table 1.

Number of tasks	Iterations to optimum		Computer time of single iteration [S]	
	Algorithm A	Algorithm B	Algorithm A	Algorithm B
10	65	12	1.5	0.9
30	250	39	3.5	3.1
60	500	88	6.2	7.3
100	1000	151	10.1	18.2

It proceeds from Table 1 that using the B-algorithm one reaches the optimum faster within the scope of studied examples. However, it seems that as the size (number of tasks) of the problem increases, the performance of A-algorithm relatively improves, but for both algorithms, the time used to reach the optimum permits their practical application to problems of up to 100 tasks.

Concluding, one should state that the B-algorithm is better for the case of homogeneous processors and may be used in parctice.

Abstract

This paper deals with deterministic problems of scheduling n preemptable tasks on m parallel processors. The structure of the set of tasks is given in the form of an activity network (i.e. a directed, acyclic graph with only one origin and only one terminal) and the minimizing of the schedule length is the performance measure. The cases of identical as well as heterogeneous processors are considered. The problem of obtaining the minimal schedule lenght is reduced to a linear programm ing problem. In order to provide facilities for solving problems of a practical size, the special procedure proposed here considerably reduces computer storage requirements. For the case of identical processors two approaches for solving the problem have been compared.

INSTITUTE OF CONTROL ENGINEERING,
TECHNICAL UNIVERSITY OF POZNAŃ
POZNAŃ, POLAND

References

[1] BŁAŻEWICZ, J., W. CELLARY, J. WĘGLARZ, Scheduling preemptable tasks on heterogeneous processors (submitted for publication).
[2] BŁAŻEWICZ, J., W. CELLARY, J. WĘGLARZ, Some computational problems of scheduling dependent and preemptable tasks to minimize schedule length, *Found. Control Engrg.*, v. 1, 1975, pp. 75—83.
[3] COFFMAN, E. G., JR. & P. J. DENNING, *Operating systems theory*, Prentice Hall, Englewood Cliffs, N. J., 1973.
[4] COFFMAN, E. G., JR. (ed), *Computer & job/shop scheduling theory*, Wiley-Interscience, 1976.
[5] COVO, A. A., Analysis of multiprocessor control organizations with partial program memory replication, *IEEE Trans. Computers*, v. C—23, 1974, pp. 113—120.
[6] GASS, S. J., *Linear programming*, McGraw—Hill, N. Y., 1969.
[7] HORVATH, E. C. & R. SETHI, Preemptive schedules for independent tasks, *Technical Report*, No 162, Computer Science Dep., Pennsylvania State Univ., 1975.
[8] MC NAUGHTON, R., Scheduling with deadlines and loss functions, *Management Sci.*, v. 6, 1959, pp. 1—12.
[9] MUNTZ, R. R. & E. G. COFFMAN, JR., Optimal preemptive scheduling on two-processor systems, *IEEE Trans. Computers*, v. C—18, 1969, pp. 1014—1020.
[10] MUNTZ, R. R. & E. G. COFFMAN, JR., Preemptive scheduling of real-time tasks on multiprocessor systems, *J. Assoc. Comput. Mach.*, v. 17, 1970, pp. 324—338.
[11] TORRES, J., Test and evaluation of computer traffic control system, *Diamond Interchange Traffic Control*, PB—224160, v. 9, 1973.

(Received Feb. 16, 1976)