stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE
(DEPARTMENT OF OPERATIONS RESEARCH)

BW 91/78 SEPTEMBER

C.N. POTTS

AN ADAPTIVE BRANCHING RULE FOR THE PERMUTATION FLOW-SHOP PROBLEM

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 90B35, 65K05

AN ADAPTIVE BRANCHING RULE FOR THE PERMUTATION FLOW-SHOP PROBLEM

C.N. POTTS
University of Keele, England

ABSTRACT

A branch and bound algorithm is presented for the permutation flow-shop problem in which the objective is to minimise the maximum completion time. A branching procedure is used in which jobs both at the beginning and at the end of the schedule have been fixed. Dominance rules are included in the algorithm. Also, during the initial stages of the algorithm, upper bounds are computed at certain nodes of the search tree. Computational results indicate that the proposed algorithm is superior to previously published algorithms.

KEY WORDS & PHRASES: *permutation flow-shop, branch and bound, adaptive branching rule, two-machine bound, newest active node search, dominance rule, upper bound, computational experience.*
NOTE: This report is not for review; it will be submitted for publication in a journal.

1. INTRODUCTION

The usual assumptions about the *permutation flow-shop problem* will be adopted. Each of n jobs is to be processed on machines 1,...,m in that order. The processing time of each job i on each machine j, denoted by p_{ij}, is given. At any time each machine can process at most one job and each job can be processed on at most one machine. Once the processing of a job on a machine has started, it must be completed without interruption. The sequence in which the jobs are to be processed is the same for each machine. The problem is to find a sequence of jobs to minimise the maximum completion time.

For m = 2, Johnson [8] has derived an algorithm requiring O(n log n) steps. However, for m = 3 it has been shown in references [6] and [12] that the problem is NP-hard. Baker [2] has shown that branch and bound methods are more efficient than enumerative methods based solely on elimination rules.

A *branch and bound algorithm* for a minimisation problem is characterised by the following:

(a) its *branching rule* which defines partitions of the set of feasible solutions into subsets;

(b) its *lower bounding rule* which provides a lower bound on the value of each solution in a subset generated by the branching rule;

(c) its *search strategy* which selects a node from which to branch.

Additional features such as *dominance rules* and *upper bounding methods* may also be present.

In Section 2 we shall outline our branching rule and in Section 3 a powerful lower bounding rule is derived. The complete algorithm is presented in Section 4 including our implementation of these rules, the search strategy, the dominance rules and the upper bounding method. Computational experience is presented in Section 5 which is followed by some concluding remarks in Section 6.

2. BRANCHING RULE

An important characteristic of most efficient branch and bound algorithms is that the decisions which have a major effect on the objective function

are made at the top of the search tree.

In almost all of the previously published algorithms for the permutation flow-shop problem [2,3,4,7,9,13,14,15,16] the same branching procedure has been used: nodes at level r of the search tree correspond to initial partial sequences in which jobs in the first r positions have been fixed. However, both Brown and Lomnicki [4] and McMahon and Burton [14] have found from computational results that in some circumstances it is more efficient to solve the inverse problem in which the processing times p_{ij} and $p_{i,m-j+1}$ are interchanged for all jobs i and all machines j such that $1 \leq j \leq m/2$ rather than the original problem. This problem inversion is equivalent to a branching procedure for the original problem in which nodes at level r of the search tree represent final partial sequences in which jobs in the last r positions have been fixed.

In this paper we are suggesting that the important decisions for some permutation flow-shop problems will involve scheduling jobs in both the first few and last few positions. Thus in the proposed algorithm each node of the search tree will correspond to an initial partial sequence σ_1 and a final partial sequence σ_2, though either σ_1 or σ_2 may be empty. The branching rule used by other researchers is a special case of our general method for which σ_2 is empty. The precise details of the proposed procedure will be given in Section 4.

3. LOWER BOUNDS

Lower bounds on the maximum completion time for all sequences beginning with the initial partial sequence σ_1 have been developed by several researchers [3,4,7,9,13,14,15,16]. The most efficient is the *two-machine bound* developed independently by Lageweg et al. [9] and Potts [16]. This will be generalised to give a lower bound on the total processing time for all sequences beginning with the initial partial sequence σ_1 and ending with the final partial sequence σ_2.

Let S_1 be the set of jobs sequenced in σ_1 and let S_2 be the set of jobs sequenced in σ_2. Also for any machine j, $C_1(\sigma_1,j)$ is defined as the minimum time to complete processing all jobs in σ_1 on machine j and $C_2(\sigma_2,j)$ is defined as the minimum time between the start of processing jobs in σ_2 on ma-

machine j and the completion of processing jobs in σ_2 on machine m. (We define $C_1(\sigma_1,j) = 0$ if $S_1 = \emptyset$ and $C_2(\sigma_2,j) = 0$ if $S_2 = \emptyset$.) Now if $S = \{1,\ldots,n\} - (S_1 \cup S_2)$ is the set of unsequenced jobs, we define

$$L_1(\sigma_1,j) = \begin{cases} C_1(\sigma_1,j) & \text{if } S_1 \neq \emptyset \\ \min_{i \in S}\{\sum_{k=1}^{j-1} p_{ik}\} & \text{if } S_1 = \emptyset \end{cases} \quad (j = 1,\ldots,m)$$

and

$$L_2(\sigma_2,j) = \begin{cases} C_2(\sigma_2,j) & \text{if } S_2 \neq \emptyset \\ \min_{i \in S}\{\sum_{k=j+1}^{m} p_{ik}\} & \text{if } S_2 = \emptyset \end{cases} \quad (j = 1,\ldots,m).$$

A lower bound is obtained by choosing a machine pair (u,v), where $1 \leq u \leq v \leq m$, and relaxing the constraint that machines u+1,...,v-1 can process only one job at a time. If $u \neq v$, a two-machine subproblem is produced in which each job i in S has a processing time p_{iu} on the first machine, a time lag of $\sum_{k=u+1}^{v-1} p_{ik}$ between the completion of processing job i on the first machine and the start of processing job i on the second machine, and a processing time p_{iv} on the second machine. An optimum sequence for this subproblem is obtained by ordering, using Johnson's rule, for a two-machine problem with processing times $\sum_{k=u}^{v-1} p_{ik}$ and $\sum_{k=u+1}^{v} p_{ik}$ for $i \in S$ [5]. Alternatively if $u = v$, a single-machine subproblem results, for which any sequence is optimum. If $T(\sigma_1,\sigma_2,u,v)$ denotes the minimum maximum completion time for the subproblem, then a lower bound is given by

$$B(\sigma_1,\sigma_2,u,v) = L_1(\sigma_1,u) + T(\sigma_1,\sigma_2,u,v) + L_2(\sigma_2,v).$$

When $S_2 = \emptyset$, $B(\sigma_1,\sigma_2,u,v)$ is identical with the lower bound used in references [9] and [16]. Computational results have indicated that it is stronger than previously published bounds. It is a generalisation of the lower bound $B(\sigma_1,\sigma_2,u,u+1)$, where $1 \leq u \leq m-1$, proposed by Nabeshima [15].

Thus a lower bound for the problem is given by specifying a set of machine pairs $W = \{(u_1,v_1),\ldots,(u_w,v_w)\}$ to give an overall lower bound defined by

$$LB(\sigma_1,\sigma_2,W) = \max\{B(\sigma_1,\sigma_2,u_1,v_1),\ldots,B(\sigma_1,\sigma_2,u_w,v_w)\}.$$

When $W = \{(1,1),\ldots,(m,m)\}$ the resulting bound is called the *machine-based bound*. The choice of the set W will be discussed in the next section.

4. THE ALGORITHM

Branching Rule

The following branching rule will be used is the proposed algorithm. Initial experiments have shown it to yield promising results. The first branching will sequence a job in position 1 while the second branching sequences a job in position n. Subsequent branchings will either be of *type 1* in which a job is added to the end of an initial partial sequence σ_1, or of *type 2* in which a job is added to the beginning of a final partial sequence σ_2. More formally, each node of the search tree can be represented by (σ_1,σ_2) where $\sigma_1 = (\sigma_1(1),\ldots,\sigma_1(s_1))$, $\sigma_2 = (\sigma_2(n-s_2+1),\ldots,\sigma_2(n))$ and $s_1 < n-s_2+1$. As before let S denote the set of unsequenced jobs. Then for $S \neq \emptyset$, a typical immediate successor of (σ_1,σ_2) is either $(\sigma_1 i,\sigma_2)$ following a type 1 branching where $\sigma_1 i = (\sigma_1(1),\ldots,\sigma_1(s_1),i)$ and $i \in S$, or $(\sigma_1,i\sigma_2)$ following a type 2 branching where $i\sigma_2 = (i,\sigma_2(n-s_2+1),\ldots,\sigma_2(n))$ and $i \in S$. The following rule will decide between type 1 and type 2 branchings during the first pass of the algorithm. Once the branching pattern has been set, it is repeated whenever backtracking is necessary. Let k_1 and k_2 denote the lowest levels of the search tree at which nodes were constructed from type 1 and type 2 branchings respectively. Also let n_1 and n_2 be the numbers of nodes at levels k_1 and k_2 which have lower bounds achieving the minimum value bound at levels k_1 and k_2 respectively. If $n_1 < n_2$ the next branching is of type 1, while if $n_1 > n_2$ the next branching is of type 2. If $n_1 = n_2$, then the next branching is of type 1 if the previous branching is of type 1; otherwise it is of type 2. Should all nodes be eliminated by dominance or upper bounds at some level of the tree whilst the branching pattern is being set, all subsequent branchings will be of the same type as the previous branching.

The branching rule used by other researchers will be referred to as B_0 while the method described above will be denoted by B_1. It is called an *adaptive branching rule* because the branching pattern is problem dependent.

Lower Bounds

The choice of the set of machine pairs used to calculate the lower bound will be discussed here.

In references [9] and [16] it was found that the sets of machine pairs $\{(1,m),\ldots,(m-1,m)\}$ and $\{(1,m),\ldots,(m,m)\}$ respectively gave good computational results. However, to ensure that our proposed bound is never less than the machine-based bound when branching rule B_1 is used, we propose the set of machine pairs

$$W_0 = \{(1,1),\ldots,(m,m),(1,m),\ldots,(m-1,m)\}.$$

One factor likely to affect the efficiency of $B(\sigma_1,\sigma_2,u,v)$ is the total processing time on machines u and v. Larger total processing times are expected to produce higher bounds. Another factor is the size of v-u: the poor results obtained by Ashour and Quraishi [1] for Nabeshima's bound indicate that $B(\sigma_1,\sigma_2,u,v)$ is likely to increase as v-u increases. With this in mind we suggest two other choices of sets of machine pairs. Firstly we define $W_1 = W_0 \cup \{(u,v)\}$ if machines u and v can be found such that $1 \le u < v < m$ and the total processing time on each of machines u and v exceeds the total processing time on all other machines; otherwise $W_1 = W_0$. Secondly we define $W_2 = W_0 - \{(u,u),(u,m)\}$ if a machine u can be found such that $(m-1)/2 \le u < m$ and the total processing time on machine u is less than the total processing time on all other machines; otherwise $W_2 = W_0$.

Search Strategy

A *newest active node search* is used which selects a node from which to branch which has the smallest lower bound amongst nodes in the most recently created subset. If there is a choice of nodes with the same minimum lower bound, then one is chosen having partial sequences which produce the smallest sum of idle times on all machines. This last tie-splitting rule is a special feature of the algorithm designed to help generate a good solution quickly. Initial experiments have shown it to be effective.

Dominance

If it can be shown that an optimum solution can always be generated without branching from a particular node, then that node is dominated and can be eliminated. Dominance rules usually specify whether a node can be eliminated before its lower bound is calculated. Clearly, dominance rules are particularly useful when a node can be eliminated which has a lower bound that is less than the optimum solution. The dominance rule developed by Szwarc [18] for the permutation flow-shop problem will be used here.

Using the notation of the previous section, let $i,j \in S$ be any two unsequenced jobs. We now define

$$\Delta_{1k} = C_1(\sigma_1 ij,k) - C_1(\sigma_1 j,k) \quad (k = 1,\ldots,m)$$

and

$$\Delta_{2k} = C_2(ji\sigma_2,k) - C_2(j\sigma_2,k) \quad (k = 1,\ldots,m).$$

Then we have the following dominance rules. If

$$\Delta_{1,k-1} \leq \Delta_{1k} \leq p_{ik} \quad \text{for } k = 2,\ldots,m, \tag{1}$$

then $\sigma_1 ij$ dominates $\sigma_1 j$. Also if

$$\Delta_{2k} \leq \Delta_{2,k-1} \leq p_{i,k-1} \quad \text{for } k = 2,\ldots,m, \tag{2}$$

then $ji\sigma_2$ dominates $j\sigma_2$. The application of these rules is limited because for (1) to hold we must have

$$p_{i1} \leq p_{ik} \quad \text{for } k = 2,\ldots,m,$$

and for (2) to hold we must have

$$p_{im} \leq p_{ik} \quad \text{for } k = 1,\ldots,m-1.$$

Baker's implementation, in which $\sigma_1 ij$ and $ji\sigma_2$ are not used to eliminate any partial sequence once $\sigma_1 i$ and $i\sigma_2$ have themselves been eliminated, will be adopted.

Upper Bounds

It is well-known that computation can be reduced by using a heuristic method to find a good solution to act as an upper bound on the maximum completion time prior to the application of a branch and bound algorithm. We propose here to calculate upper bounds at certain nodes of the search tree during the application of the algorithm. With either approach, the minimum number of nodes can be reduced from n(n+1)/2 to n.

At each node of the search tree a machine pair (u,v) can be found which provides the lower bound for that node. Also, there exists a corresponding ordering of the unsequenced jobs that is used in calculating the bound. This provides us with a sequence of jobs for which the maximum completion time yields an upper bound. This upper bound is calculated for a node immediately prior to branching from it, provided that this node was created at the previous branching so that the appropriate machine pair does not have to be stored or recalculated. To avoid unnecessary calculation of upper bounds in cases where most of the computational effort is spent on proving the optimality of a certain solution, it was decided to apply the upper bounding procedure to only the first n such nodes.

Algorithm Representation

It can be seen from the specifications above that each algorithm to be considered can be represented by (W,BR,DOM,UB), where

W = W_0, W_1 or W_2 describes the set of machine pairs to be used in the calculation of the lower bound;

BR = B_0 or B_1 describes the branching rule;

DOM = - or D if the dominance rule is not used or used respectively;

UB = - or U if the upper bounds are not used or used respectively.

5. COMPUTATIONAL EXPERIENCE

The problems used to compare the algorithms contained *random* problems, problems with *correlation* between the processing times of each job, problems for which the processing times of each job have a *positive trend* and final-

ly problems with correlation and a positive trend for the processing times of each job. We shall denote these *problem classes* by R, C, T and CT respectively. Twenty five problems of each type were generated for the n/m values 8/5, 8/7, 10/3, 10/5, 10/7, 15/5, and 20/3. The method of problem generation follows that given in reference [9]. The algorithms were coded in FORTRAN IV and run on a CDC 7600 computer. Computational results are given in Tables I, II and III. Whenever a problem was not solved after 100,000 nodes had been generated, computation was abandoned for that problem. Thus in some cases the figures given in Tables I and II will be lower bounds on average computation times and average numbers of nodes.

The first three columns of Tables I and II compare the performance of the three sets of machine pairs W_0, W_1 and W_2 using the branching rule B_0. It is seen that W_0 performs best and will be used henceforth. Column 4 shows that the effect of introducing the dominance rule is to reduce computation, which confirms the results of reference [9]. A closer examination shows that most of the saving comes from the problem class CT and, to a lesser extent, the class C. The problems from class T could usually be solved with the minimum number of nodes with or without the dominance rule. For more than three machines, the dominance rule was most ineffective when applied to the random problems. The larger average computation time for the 10/5 problems compared with the 10/7 problems in the first four columns of Tables I and II is probably a random effect which could have been eliminated if more problems had been solved.

Columns 5 and 6 of Tables I and II show the effect of using the adaptive branching rule without and with dominance. Clearly there are substantial savings in computation compared with the corresponding results in columns 1 and 4 where B_0 is used. Finally by adding our upper bounding procedure, column 7 shows that a further small reduction in computation can be achieved.

The numbers of unsolved problems for the two branching procedures, with and without dominance, are classified according to problem type in Table III. An unexpected result is observed in the last two columns of Table III where, for the 15/5 problems in class CT, more problems were unsolved when dominance was applied than without. Further examination of these problems reveals a different branching pattern for the same problem caused by

TABLE I. AVERAGE COMPUTATION TIMES*†

		Algorithm						
n	m	$(W_0,B_0,-,-)$	$(W_1,B_0,-,-)$	$(W_2,B_0,-,-)$	$(W_0,B_0,D,-)$	$(W_0,B_1,-,-)$	$(W_0,B_1,D,-)$	(W_0,B_1,D,U)
8	5	1.12	1.22	1.15	0.70	0.92	0.84	0.70
8	7	3.76	3.95	3.78	2.57	2.53	2.23	2.02
10	3	2.42	2.78	2.42	1.21	1.05	0.54	0.38
10	5	41.13	43.39	41.13	19.44	9.24	7.73	7.37
10	7	38.95	41.04	39.11	15.59	19.22	11.57	10.99
15	5	228.51	251.61	231.34	224.35	120.97	101.99	99.89
20	3	99.96	111.18	99.96	65.89	73.94	24.10	22.33

* Lower bounds on the average when there are unsolved problems.

† Times are in hundredths of a CPU second.

TABLE II. AVERAGE NUMBERS OF NODES*

		Algorithm						
n	m	$(W_0,B_0,-,-)$	$(W_1,B_0,-,-)$	$(W_2,B_0,-,-)$	$(W_0,B_0,D,-)$	$(W_0,B_1,-,-)$	$(W_0,B_1,D,-)$	(W_0,B_1,D,U)
8	5	152	150	156	80	111	92	76
8	7	423	410	427	253	263	204	189
10	3	521	520	525	227	179	90	53
10	5	6282	6049	6382	2538	1273	971	941
10	7	4506	4465	4525	1504	2123	1117	1079
15	5	31791	31792	32252	26966	15690	11997	11884
20	3	16802	16801	16932	10302	10928	3883	3594

* Lower bounds on the average when there are unsolved problems.

the elimination of nodes by dominance. However, this anomaly can be easily overcome by not discarding dominated nodes until the branching pattern has been set.

It is interesting to note how our branching rule adapts itself to the four classes of problem. These results were observed when dominance was not used. For random problems small groups of consecutive type 1 and type 2 branchings occur towards the top of the search tree. This indicates that the jobs sequenced in the initial few and final few positions may largely determine the total processing time. For problems in class C, branchings tend either to be all of type 1 or all of type 2 in the top half of the search tree. The jobs with large processing times, which are sequenced in the middle positions in an optimum sequence, provide the main contributions to the total processing time. As these jobs cannot be sequenced under B_0 or B_1 until approximately one half of the other jobs have been scheduled, the difficulty in solving such problems is hardly surprising. For problems in class T the branchings tend to be of type 1. It appears that only the first few jobs affect the total processing time here. Finally for the CT class, the initial branchings all tend to be of type 2. The jobs with large processing times are sequenced towards the end for these problems. As these correlated problems with a positive trend are best solved by type 2 branchings, the inverse problem ought to be solved when branching rule B_0 is adopted. This contradicts results obtained by other researchers [4,14] who did not test their algorithms on problems in class CT.

Finally the most efficient of our algorithms (W_0,B_1,D,U) was applied to groups of problems with n/m values of 50/3, 50/4, 50/5, 100/3 and 100/4. Each group contained 20 problems generated in the same way as the problems for the previous tests. The results are given in Table IV.

As expected, increasing the number of jobs or machines increases computation. However, there is some evidence that as the number of jobs increases the numbers of unsolved problems decrease. This may be because for larger numbers of jobs there is more likely to be a job with "suitable" processing times to be sequenced in a given position.

Of the 22 unsolved problems, 2 were in class R, 16 were in class C and 4 were in class CT. Thus the correlated problems appear the most challenging, confirming the results of reference [9].

TABLE III. NUMBERS OF UNSOLVED PROBLEMS*

n	m	Problem Class	Algorithm $(W_0,B_0,-,-)$	$(W_0,B_0,D,-)$	$(W_0,B_1,-,-)$	$(W_0,B_1,D,-)$
10	5	R	0	0	0	0
		C	0	0	0	0
		T	0	0	0	0
		CT	2	0	0	0
10	7	R	0	0	0	0
		C	0	0	0	0
		T	0	0	0	0
		CT	1	0	0	0
15	5	R	9	9	2	2
		C	6	3	7	1
		T	0	0	0	0
		CT	14	10	3	5
20	3	R	6	5	0	0
		C	4	1	5	1
		T	0	0	0	0
		CT	5	3	3	1

* No unsolved problems for the n/m values 8/5, 8/7, and 10/3.

TABLE IV. COMPUTATIONAL RESULTS FOR LARGER PROBLEMS

n	m	Average Computational Time*†	Average Number of Nodes*	Number of Unsolved Problems
50	3	138.80	10452	2
50	4	386.20	30549	6
50	5	850.00	44367	8
100	3	158.25	7487	1
100	4	1004.05	30934	5

* Lower bound on the average when there are unsolved problems.

† Times are in hundredths of a CPU second.

6. CONCLUDING REMARKS

The use of the adaptive branching rule enables computation to be reduced by over 50% for some larger problems. It seems likely that a similar branching rule could be effectively applies to other machine scheduling problems such as permutation flow-shop problems with different objectives and job-shop problems.

In spite of the improved results achieved by our algorithm, it seems that a different approach is needed for correlated problems. An approach based on selecting certain pairs of jobs and deciding, at the top of the search tree, an ordering between the two jobs of each pair seems worth investigating. Such an algorithm has been applied to the job-shop problem by Lageweg et al. [10]. Improving the lower bounds, perhaps using subgradient optimisation, should also yield a more efficient algorithm.

ACKNOWLEDGEMENT

The author is grateful to the Mathematisch Centrum, Amsterdam and The Royal Society for helping to finance a visit to the Mathematisch Centrum where this research was completed. Also useful advice by J.K. Lenstra on the presentation of this paper is gratefully acknowledged.

REFERENCES

1. S. Ashour and M.N. Quraishi, "Investigation of Various Bounding Procedures for Production Scheduling Problems", Internat. J. Production Res. 7, 249-252 (1969).
2. K.R. Baker, "A Comparative Study of Flow-Shop Algorithms", Operations Res. 23, 62-73 (1975).
3. P.F. Bestwick and N.A.J. Hastings, "A New Bound for Machine Scheduling", Operational Res. Quart. 27, 479-487 (1976).
4. A.P.G. Brown and Z.A. Lomnicki, "Some Applications of the 'Branch-and Bound' Algorithm to the Machine Scheduling Problem", Operational Res. Quart. 17, 173-186 (1966).
5. R.W. Conway, W.L. Maxwell and L.W. Miller, "Theory of Scheduling",

Addison-Wesley, Reading, Mass. 1967.

6. M.R. Garey, D.S. Johnson and R. Sethi, "The Complexity of Flowshop and Jobshop Scheduling", Math. Operations Res. 1, 117-129 (1976).
7. E. Ignall and L. Schrage, "Applications of the Branch-and-Bound Technique to Some Flow-Shop Scheduling Problems", Operations Res. 13, 400-412 (1965).
8. S.M. Johnson, "Optimal Two- and Three-Stage Production Schedules with Setup Times Included", Naval Res. Log. Quart. 1, 61-68 (1954).
9. B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, "A General Bounding Scheme for the Permutation Flow-Shop Problem", Operations Res. 26, 53-67 (1978).
10. B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, "Job-Shop Scheduling by Implicit Enumeration", Management Sci. 24, 441-450 (1977).
11. J.K. Lenstra, "Sequencing by Enumerative Methods", Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam 1977.
12. J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, "Complexity of Machine Scheduling Problems", Ann. Discrete Math. 1, 343-362 (1977).
13. Z.A. Lomnicki, "A 'Branch-and Bound' Algorithm for the Exact Solution of the Three-Machine Scheduling Problem", Operational Res. Quart. 16, 89-100 (1965).
14. G.B. McMahon and P.G. Burton, "Flow-Shop Scheduling with the Branch-and-Bound Method", Operations Res. 15, 473-481 (1967).
15. I. Nabeshima, "On the Bound of Makespans and Its Application in M Machine Scheduling Problem", J. Operations Res. Soc. Japan 9, 98-136 (1967).
16. C.N. Potts, "The Job-Machine Scheduling Problem", Ph.D. thesis, University of Birmingham, 1974.
17. A.H.G. Rinnooy Kan, "Machine Scheduling Problems: Classification, Complexity and Computations", Nijhoff, The Hague 1976.
18. W. Szwarc, "Elimination Methods in the $m \times n$ Sequencing Problem", Naval Res. Log. Quart. 18, 295-305 (1971).

ONTVANGEN 3 0 OKT. 1978