stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 52/75 AUGUST

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN

MINIMIZING MAXIMUM LATENESS ON ONE MACHINE: COMPUTATIONAL EXPERIENCE AND SOME APPLICATIONS

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), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS (MOS) subject classification scheme (1970): 90B35

MINIMIZING MAXIMUM LATENESS ON ONE MACHINE: COMPUTATIONAL EXPERIENCE AND SOME APPLICATIONS

B.J. LAGEWEG, J.K. LENSTRA
Mathematisch Centrum, Amsterdam

A.H.G. RINNOOY KAN
Graduate School of Management, Delft

ABSTRACT

We consider the problem of scheduling jobs on a single machine subject to given release dates and precedence constraints, in order to minimize maximum lateness. The algorithms of Baker and Su (*Naval Res. Logist. Quart.* 21(1974) 171-176) and of McMahon and Florian (*Operations Res.* 23(1975)475-482) for the problem without precedence constraints are extended to the general case. An extensive computational comparison establishes the superiority of the latter algorithm. We describe applications to the theory of job-shop scheduling and to a practical scheduling situation.

KEY WORDS & PHRASES: *one-machine scheduling, release dates, due dates, precedence constraints, maximum lateness, job-shop scheduling, practical scheduling*

1. INTRODUCTION

We consider the following problem. Suppose we have n *jobs* $J_1,\ldots,J_n$, to be processed on a single *machine* which can handle only one job at a time. Job J_i $(i = 1,\ldots,n)$ is available for processing at its *release date* r_i, requires an uninterrupted *processing time* of p_i time units and should ideally be completed by its *due date* d_i. Certain *precedence constraints* define a partial ordering $<$ between the jobs; "$J_i < J_j$" means that J_j cannot start before the completion of J_i. The index sets $B_i = \{j | J_j < J_i\}$ and $A_i = \{j | J_i < J_j\}$ indicate the jobs which are constrained to come before and after J_i respectively. Given a feasible processing order of the jobs, we can compute for J_i $(i = 1,\ldots,n)$ a *starting time* $S_i \geq r_i$, a *completion time* $C_i = S_i + p_i$ with $C_i \leq S_j$ for all $j \in A_i$, and a *lateness* $L_i = C_i - d_i$. We want to find a schedule that minimizes the maximum lateness $L_{max} = \max_i\{L_i\}$.

A number of special cases of this problem in which all r_i, p_i or d_i are equal is studied in section 2. In sections 3 and 4, two branch-and-bound algorithms developed for the problem without precedence constraints are described and extended to the general case. Extensive computational experience is reported in section 5. In sections 6 and 7 we discuss two applications of the problem; one arises in the theoretical context of job-shop scheduling, the other occurred in a practical scheduling situation. Concluding remarks are contained in section 8.

We will present an ALGOL-like description of several algorithms; the operation "$:\in$" in the statement "s:$\in$ S" is defined to mean that s becomes an arbitrary element of S.

2. SPECIAL CASES

The problem to be considered is defined by n integer triples (r_i,p_i,d_i) and precedence constraints $<$.

To stress the *symmetry* inherent to the problem, it is useful to describe it in an alternative way. Let M_1 and M_3 be *non-bottleneck* machines of infinite capacity and M_2 a *bottleneck* machine of capacity one. Job J_i $(i = 1,\ldots,n)$ has to visit M_1,M_2,M_3 in that order and has to spend

- a *head* r_i on M_1 from 0 to r_i;
- a *body* p_i on M_2 from $S_i \geq r_i$ to $C_i = S_i+p_i$ with $C_i \leq S_j$ for all $j \in A_i$;
- a *tail* $q_i = K-d_i$ (for some constant $K \geq \max_i\{d_i\}$) on M_3 from C_i to $L_i' = C_i+q_i = L_i+K$.

We want to minimize the total processing time $L'_{max} = \max_i\{L_i'\} = L_{max}+K$.

The problem, now defined by n triples (r_i,p_i,q_i) and $<$, is clearly equivalent to its *inverse* problem defined by (q_i,p_i,r_i) and $<'$ with $J_i <' J_j$ if $J_j < J_i$; an optimal schedule for one problem can be reversed to obtain an optimal schedule for the other problem, with the same solution value.

Let us first assume that $A_i = B_i = \emptyset$ for all i.

If all r_i are equal, an optimal schedule is provided by *Jackson's rule* [12]: L_{max} is minimized by ordering the jobs according to nondecreasing d_i.

If all d_i are equal, the problem is similarly solved by ordering the jobs according to nondecreasing r_i. This result can be interpreted as a consequence of the symmetry discussed above.

If all p_i are equal, such a simple solution method is usually not available, unless $p_i = 1$ for all i. In the latter situation, algorithm J below involving repeated application of Jackson's rule produces an optimal schedule [1].

```
procedure algorithm J (n,r,q,S,C);
begin   local N,N',t,i;
        N:= {1,...,n}; t:= 0;
        while N ≠ ∅ do
        begin   t:= max{t,min{r_j | j ∈ N}};
                N':= {j | j ∈ N, r_j ≤ t};
                i:∈ {j | j ∈ N', q_j = max{q_k | k ∈ N'}};
                N:= N-{i}; S_i:= t; C_i:= t:= t+1
        end
end.
```

The proof of this result is straightforward and depends on the fact that no job can become available during the processing of another one, so that it is never advantageous to postpone processing the selected job J_i. This argument does not apply if $p_i = p$ for all i and p does not divide all r_i; e.g., if $n = p = 2$, $r_1 = q_1 = 0$, $r_2 = 1$, $q_2 = 2$, postponing J_1 is clearly advantageous. However, algorithm J does solve the general problem if we allow *job splitting* (i.e., interruptions in the processing of a job); in this case we can interpret job J_i as p_i jobs with heads r_i, bodies 1 and tails q_i (cf. [11]).

Let us now examine the introduction of precedence constraints in the problems discussed so far. As a general principle, note that we may set

$$r_i := \max\{r_i, \max\{r_j+p_j \mid j \in B_i\}\}$$
$$q_i := \max\{q_i, \max\{p_j+q_j \mid j \in A_i\}\}$$

because in every feasible schedule $S_i \geq C_j \geq r_j+p_j$ for all $j \in B_i$ and $L_j' \geq C_i+p_j+q_j$ for all $j \in A_i$. Hence, if $J_i < J_j$, we may assume that $r_i < r_i+p_i \leq r_j$ and $q_i \geq q_j+p_j > q_j$.

It follows that the case in which all d_i are equal is again solved by ordering the jobs according to nondecreasing r_i. Such an ordering will respect all precedence constraints in view of the preceding argument.

If we apply this method to the inverse problem to solve the case in which all r_i are equal, the resulting algorithm can be interpreted as a special case of Lawler's more general algorithm to minimize $\max_i\{c_i(C_i)\}$ for arbitrary nondecreasing cost functions c_i [14].

A similar observation can be made with respect to the case that $p_i = 1$ for all i. Algorithm J will produce a schedule respecting the precedence constraints.

So far all the methods presented have been *good* or *efficient* algorithms in the by now conventional sense that their number of steps is bounded by a polynomial in n [8]. Such a method is unlikely to exist for the case in which r_i, p_i and q_i may assume arbitrary values and $A_i = B_i = \emptyset$ for all i; in [3], this problem was proved to be *NP-complete*, which implies that an efficient method for its solution would yield good algorithms for all other

NP-complete problems as well. Because many notorious combinatorial problems such as the travelling salesman problem, job-shop scheduling and graph coloring are NP-complete, the NP-completeness of the problem without precedence constraints serves as a formal justification to use enumerative solution methods such as *branch-and-bound*. Algorithms of this type have been proposed by Dessouky and Margenthaler [7], Bratley, Florian and Robillard [2], Baker and Su [1] and McMahon and Florian [17]. The first of these algorithms is not stated very clearly; the second one is surpassed by the fourth one both in elegance and efficiency [17]. In the following two sections, the algorithms from [1] and [17] will be described and extended to the general case.

3. THE ALGORITHM OF BAKER AND SU

The branch-and-bound algorithm to be discussed now has been developed by Baker and Su [1] for the problem without precedence constraints. It will be referred to as algorithm BS.

The *branching rule* generates all *active* schedules [6] according to algorithm AS below.

```
procedure algorithm AS (n,r,p,S,C);
begin   local i;

        procedure node(N,t);
        if N = ∅ then comment an active schedule has been generated else
        begin   local N';
                N':= {j | j ∈ N, r_j < min{max{t,r_k}+p_k | k ∈ N}};
                while N' ≠ ∅ do
                begin   i:∈ N';
                        N':= N'-{i}; S_i:= max{t,r_i}; C_i:= S_i+p_i;
                        node(N-{i},C_i)
                end
        end;
```

node({1,...,n},0)

end.

At the ℓ-th level of the recursion, jobs are scheduled in the ℓ-th position. If the first assignment to N' is replaced by N' := N, all n! schedules are generated. By means of the current assignment, only active schedules are generated; if $r_j \geq \max\{t,r_k\}+p_k$ for some $j,k \in N$, J_j is no candidate for the next position in the partial schedule since it can be preceded by J_k without being postponed.

The *bounding rule* is based on the observation that the value of an optimal schedule will not increase if we allow job splitting. A *lower bound* on all possible completions of a partial schedule $(J_{\pi(1)},\ldots,J_{\pi(\ell)})$ is produced by the use of algorithm J to schedule the remaining jobs from $C_{\pi(\ell)}$ onwards while allowing job splitting. If no job splitting occurs, this particular completion is an optimal one, and the value of the complete solution is an *upper bound* on the value of an optimal solution. A partial schedule can be *eliminated* if its lower bound is not smaller than the global upper bound.

The branch-and-bound algorithm is now completely described if we specify a *search strategy* indicating which partial schedule will be chosen for further examination (cf. [16]). In [1], a *jumptrack* scheme was used, selecting a partial schedule with minimum lower bound. We implemented the recursive *backtrack* scheme of algorithm AS, selecting the unscheduled jobs in the order in which they appear in the solution, produced by algorithm J. Experiments in which these descendant nodes were chosen in order of non-decreasing lower bounds showed a 50-60 percent increase in solution time.

The above algorithm can easily be adjusted to take precedence constraints $<$ into account. As noted previously, they are automatically respected during the lower bound calculation and the only necessary change is a replacement of the first assignment to N' by

$$N' := \{j \mid j \in N,\ B_j \cap N = \emptyset,\ r_j < \min\{\max\{t,r_k\}+p_k \mid k \in N,\ B_k \cap N = \emptyset\}\}.$$

Algorithm BS is fairly straightforward and its general principles can be extended to other NP-complete sequencing problems with non-equal release dates.

4. THE ALGORITHM OF McMAHON AND FLORIAN

A more sophisticated branch-and-bound algorithm for the problem without precedence constraints is due to McMahon and Florian [17]. Algorithm MF is based on algorithm S below, a heuristic method suggested by Schrage [20] for generating a good solution.

```
procedure algorithm S (n,r,p,q,S,C);
begin   local N,N',t,i;
        N:= {1,...,n}; t:= 0;
        while N ≠ ∅ do
        begin   t:= max{t,min{r_j | j ∈ N}};
                N':= {j | j ∈ N, r_j ≤ t};
                i:∈ {j | j ∈ N', p_j = max{p_k | k ∈ N', q_k = max{q_ℓ | ℓ ∈ N'}}};
                N:= N-{i}; S_i:= t; C_i:= t:= t+p_i
        end
end.
```

The schedule $(J_{\pi(1)},\ldots,J_{\pi(n)})$ produced by algorithm S can be decomposed into blocks. $J_{\pi(h)}$ is the *last job in a block* if $C_{\pi(h)} \leq r_{\pi(i)}$ for $i = h+1,\ldots,n$, i.e., if no job is delayed when $J_{\pi(h)}$ is completed. A set of jobs $\{J_{\pi(g)},\ldots,J_{\pi(h)}\}$ forms a *block* if

(a) $g = 1$ or $J_{\pi(g-1)}$ is the last job in a block;

(b) $J_{\pi(i)}$ is not the last job in a block, for $i = g,\ldots,h-1$;

(c) $J_{\pi(h)}$ is the last job in a block.

It follows that $J_{\pi(g)}$ is the *first job in a block* if $S_{\pi(g)} = r_{\pi(g)} \leq r_{\pi(i)}$ for $i = g+1,\ldots,n$.

With respect to J_i in block $\{J_{\pi(g)},\ldots,J_{\pi(h)}\}$, we define $P_i = \{j \mid S_{\pi(g)} \leq S_j \leq S_i\}$, $q_i^* = \min\{q_j \mid j \in P_i\}$ and $Q_i = \{j \mid j \in P_i,\ q_j = q_i^*\}$. We claim that lower bounds on the value of an optimal schedule are given by

$$LB_i' = r_i+p_i+q_i,$$

$$LB_i'' = \begin{cases} C_i+q_i^* & \text{if } i \in Q_i, \\ C_i+q_i^*+1 & \text{if } i \notin Q_i. \end{cases}$$

LB_i' requires no comment, but the justification of LB_i'' is actually rather subtle. Defining C_{ji} as the minimum completion time of J_j if this job is scheduled as the last one of $\{J_k | k \in P_i\}$, we note that $C_{ji} \geq C_{ii} = C_i$ for all $j \in P_i$. A valid lower bound is now given by

$$\min\{C_{ji}+q_j | j \in P_i\}.$$

In the case that $i \in Q_i$, it is obvious that

$$C_{ji}+q_j \geq C_{ii}+q_i = C_i+q_i^* \quad \text{for } j \in P_i. \tag{1}$$

Suppose now that $i \notin Q_i$. We have

$$C_{ji}+q_j \geq C_i+q_i^*+1 \qquad \text{for } j \in P_i-Q_i. \tag{2}$$

Consider next any $j \in Q_i$. J_j is not the last job of $\{J_k | k \in P_i\}$. If we put it last, a gap of at least one unit idle time is unavoidable, unless a J_k following J_j can be moved forward to start at S_j. From algorithm S we know that, if $r_k \leq S_j < S_k$ for some J_k, then $k \in Q_i$ and $p_k \leq p_j$. Thus, a gap threatens to occur between S_k and S_k+1. Repeating this argument as often as necessary, we conclude that $C_{ji} \geq C_i+1$, and therefore

$$C_{ji}+q_j \geq C_i+q_i^*+1 \qquad \text{for } j \in Q_i. \tag{3}$$

Inequalities (1), (2) and (3) establish the validity of LB_i''.

At every node of the search tree, application of algorithm S yields a complete solution $(J_{\pi(1)},\ldots,J_{\pi(n)})$ with value L'_{max} and a *lower bound* $LB = \max_i\{\max\{LB_i', LB_i''\}\}$. We may decrease the *upper bound* UB on the value of an optimal solution by setting $UB := \min\{UB, L'_{max}\}$. If $LB \geq UB$, the node is *eliminated*; else, we apply the *branching rule* described below.

Let the *critical job* J_i be defined as the first job in the schedule with $C_i+q_i = L'_{max}$. The schedule can only be improved if C_i can somehow be reduced. The set of solutions corresponding to the current node can now be partitioned into disjoint subsets, each characterized by the job J_j which is to be scheduled last of $\{J_k | k \in P_i\}$. However, jobs J_j with $j \in P_i$, $q_j \geq q_i-L'_{max}+UB$ need not to be considered, since in that case $C_{ji}+q_j \geq$ $\geq C_i+q_i-L'_{max}+UB = UB$. Therefore, only for each J_j with $j \in P_i$, $q_j < q_i-L'_{max}+UB$ a descendant node is actually created.

We can effectively implement the precedence constraints $\{J_k < J_j | k \in P_i - \{j\}\}$ by adjusting r_j and q_k ($k \in P_i - \{j\}$) as described in section 2. During the next application of algorithm S, J_j will then be scheduled last of $\{J_k | k \in P_i\}$. To maintain disjointness at deeper levels of the tree, we would have to update r_k and q_k for $k \notin P_i$ as well in view of previous choices. This would lead to the time consuming administration of a continually changing precedence graph. Dropping the requirement of disjoint descendants, we will force J_j to follow the critical job J_i rather than the whole set $\{J_k | k \in P_i - \{j\}\}$. This can be done by putting r_j equal to any lower bound on $C_{ji} - p_i$ not less than r_i, such as $\max\{r_k + p_k | k \in P_i - \{j\}\}$, $C_i - p_j$ or simply [17] r_i. Computational experiments have shown that the choice of a specific new r_j has only a minor influence on the performance of the algorithm; in our implementation, we put $r_j := \max\{r_i + p_i, C_i - p_j\}$.

The *search strategy* used in [17] is of the jumptrack type, selecting a node with minimum lower bound. Again, our implementation is of the recursive backtrack type, choosing the descendant nodes in the reverse of the order in which the corresponding jobs J_j appear in the solution produced by algorithm S.

Algorithm MF is easily adapted to deal with given precedence constraints $<$. Since we may assume that $r_i < r_j$ and $q_i > q_j$ if $J_i < J_j$, they are respected by algorithm S. Obviously, the lower bound remains a valid one. With respect to the branching rule, descendant nodes have to be created only for jobs J_j with $j \in P_i$, $q_j < q_i - L'_{max} + UB$, $A_j \cap P_i = \emptyset$. We could branch by adding the precedence constraints $\{J_k < J_j | k \in P_i - \{j\}\}$; many heads and tails would then have to be adjusted. If, however, we drop the requirement of disjoint descendants and aim to preserve only the original precedence constraints, we may just as well restrict ourselves to adjust r_j in the way described above and update r_k for all $k \in A_j$. Since the tails still reflect the original precedence constraints, new solutions produced by algorithm S will respect those. Again, more extensive adjustments turn out to result in additional computing time.

5. COMPUTATIONAL EXPERIENCE

Algorithms BS and MF were coded in ALGOL 60 and run on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam.

For each test problem with n jobs, 3n integer data r_i, p_i, q_i were generated from uniform distributions between 1 and r_{max}, p_{max} and q_{max} respectively. Here, $r_{max} = R.p_{max}$ and $q_{max} = Q.p_{max}$. In the precedence graph, each arc (J_i, J_j) with $i < j$ was included with probability P. Table 1 shows the values of (n, p_{max}, R, Q, P) during our experiments; the values used in previous tests are also given. For each combination of values with $R \leq Q$ five problems were generated; inversion of these problems provided test problems with $R \geq Q$ (cf. section 2). Significant and systematic differences between the solution times of a problem and its inverse would indicate advantages to be gained from problem inversion.

parameter	[1]	[17]	h.l.
n	10,20,30	20,50	20,40,80
p_{max}	2000/n	25	50
R	.5n	.5n,2n	.5,2,.5n,2n
Q	.75n,.875n,n*	.4,1,3	.5,2,.5n,2n
P	0	0	0,.05,.15,.45

Table 1 Values of parameters of test problems.

* In this case, the q_i are not distributed uniformly.

Tables 2 and 3 show the computational results for problems without precedence constraints, i.e., with P = 0. Algorithm BS solves 294 out of 300 problems with up to 80 jobs within the time limit of ten seconds. The limit is never exceeded for problems of the type on which the method has been tested previously [1]. Inspection of the results revealed no obvious rule according to which problem inversion might take place and hence this additional feature was not incorporated into algorithm BS. Even better results were obtained with algorithm MF. It turns out that this method has been tested previously [17] on the very easiest types of problems. In general,

algorithm MF performs especially well on problems with R > Q. Accordingly, we also tested algorithm FM, which inverts a problem if $\max_i\{r_i\}-\min_i\{r_i\} < \max_i\{q_i\}-\min_i\{q_i\}$ before applying algorithm MF. The remarkable quality of algorithm FM is clear from Tables 2 and 3.

Table 4 shows the effect of precedence constraints, which was investigated only with respect to algorithms MF and FM. For problems with P ≥ .15, most of the solution time is spent on adjusting the r_i and q_i in accordance with the precedence constraints, as described in section 2; this takes .06 seconds for n = 20, P = .15 and .70 for n = 80, P = .45. For each positive value of P which we tested, the median number of generated nodes is equal to one; for P = .45 branching never occurs. Inversion according to the rule given above leads to some improvement, albeit not so spectacular as in the case without precedence constraints.

n	P	median solution time			maximum solution time		
		alg. BS	alg. MF	alg. FM	alg. BS	alg. MF	alg. FM
20	0	.05	.02	.03	>10:2	.99	.11
40	0	.09	.06	.06	1.09	>10:1	.17
80	0	.23	.16	.15	>10:4	>10:3	.57

Table 2 Computational results for P = 0: a survey.

n = 80	maximum solution time											
P = 0	algorithm BS				algorithm MF				algorithm FM			
R↓ Q→	.5	2	.5n	2n	.5	2	.5n	2n	.5	2	.5n	2n
.5	.26	.25	5.54	5.75	.19	.21	1.64	>10:1	.19	.25	.15	.14
2	.25	.25	>10:1	4.84	.19	.18	>10:1	>10:1	.25	.22	.19	.16
.5n	.24	.27	3.43	3.67	.19	.17	.33	.47	.20	.25	.57	.17
2n	>10:1	>10:2	3.60	2.54	.10	.12	.52	.11	.08	.11	.49	.17
	.10	.11	2.51	2.55	.09	.07	.13	.13	.09	.08	.12	.19

Table 3 Computational results for P = 0: the influence of R and Q.

n	P	median solution time		maximum solution time	
		algorithm MF	algorithm FM	algorithm MF	algorithm FM
20	0	.02	.03	.99	.11
	.05	.06	.05	.41	.43
	.15	.07	.07	.14	.15
	.45	.07	.08	.12	.11
80	0	.16	.15	>10:3	.57
	.05	.36	.33	>10:6	>10:4
	.15	.47	.42	.85	.57
	.45	.73	.75	.81	.80

Table 4 Computational results: the influence of P.

solution times : CPU seconds on a Control Data Cyber 73-28; each entry in Table 2(3,4) represents 100(5,100) test problems.

>ℓ:k : the time limit ℓ is exceeded k times.

algorithm BS : see section 3.

algorithm MF : see section 4.

algorithm FM : algorithm MF with problem inversion if $\max_i\{r_i\}-\min_i\{r_i\} < \max_i\{q_i\}-\min_i\{q_i\}$.

n : number of jobs.

R : relative range of r_i.

Q : relative range of q_i.

P : expected density of precedence graph.

Legend to Tables 2, 3 and 4

Summing up our computational experience we conclude that, NP-complete though the problem may be, algorithms BS, MF and FM (especially the latter one) are able to solve problems of reasonable size fairly quickly. In view of the applications which are to be discussed in the following sections, this is a hopeful result.

6. A THEORETICAL APPLICATION

The one-machine problem can find application in the theoretical context of the *general job-shop scheduling problem*. This classical combinatorial problem can be formulated as follows [6;18].

Suppose we have n *jobs* $J_1,\ldots,J_n$ and m *machines* $M_1,\ldots,M_m$ which can handle at most one job at a time. Job J_i $(i = 1,\ldots,n)$ consists of a sequence of n_i *operations* O_r $(r = \sum_{j=1}^{i-1} n_j+1,\ldots,\sum_{j=1}^{i} n_j)$ each of which corresponds to the processing of job J_i on machine $\mu(O_r)$ during an uninterrupted *processing time* of p_r time units. We seek to find a processing order on each machine such that the maximum completion time is minimized.

The above problem is conveniently represented by means of a *disjunctive graph* $G = (V,C\cup D)$ [19] where

- V is the set of *vertices*, representing the operations, including fictitious initial and final operations:
  $V = \{0,1,\ldots,\sum_{j=1}^{n} n_j,*\}$;
- C is the set of directed *conjunctive arcs*, representing the given machine orders of the jobs:
  $C = \{(0,\sum_{j=1}^{i-1} n_j+1) \mid i = 1,\ldots,n\} \cup$
  $\{(r,r+1) \mid r = \sum_{j=1}^{i-1} n_j+1,\ldots,\sum_{j=1}^{i} n_j-1,\ i = 1,\ldots,n\} \cup$
  $\{(\sum_{j=1}^{i} n_j,*) \mid i = 1,\ldots,n\}$;
- D is the set of directed *disjunctive arcs*, representing the possible processing orders on the machines:
  $D = \{(r,s) \mid \mu(O_r) = \mu(O_s)\}$;
- a *weight* p_r is attached to each vertex r, with $p_0 = p_* = 0$.

The disjunctive graph for a problem with n = m = 3 is drawn in Figure 1.

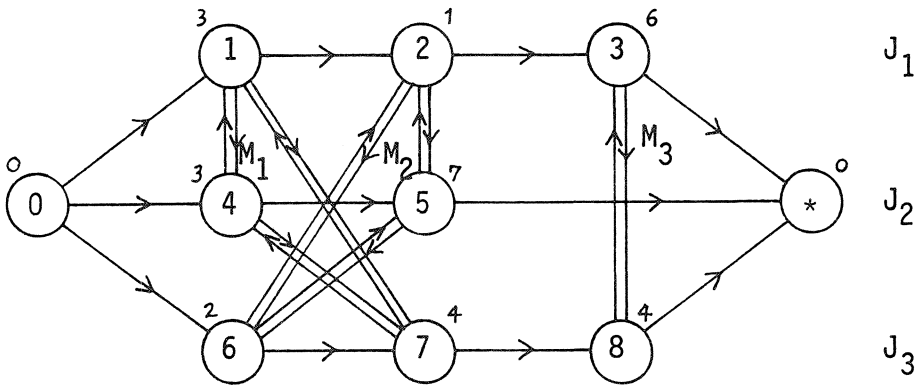

Figure 1 Example of a disjunctive graph $G = (V,C\cup D)$.

Processing orders on the machines can be defined by a subset $D \subset \mathcal{D}$ such that $(r,s) \in D$ if and only if $(s,r) \in \mathcal{D}$-D. D contains the *chosen* or *settled* arcs: if $(r,s) \in D$, then O_r precedes O_s on their common machine. The resulting schedule is *feasible* if the graph $G(D) = (V, C \cup D)$ contains no directed cycles. The value of such a schedule is given by the weight of the maximum-weight path (also called "longest" or "critical" path) in G(D). As to our example, the graph G(D) corresponding to processing orders (O_1,O_4,O_7) on M_1, (O_6,O_2,O_5) on M_2 and (O_3,O_8) on M_3 is drawn in Figure 2; the value of the schedule is equal to 18.

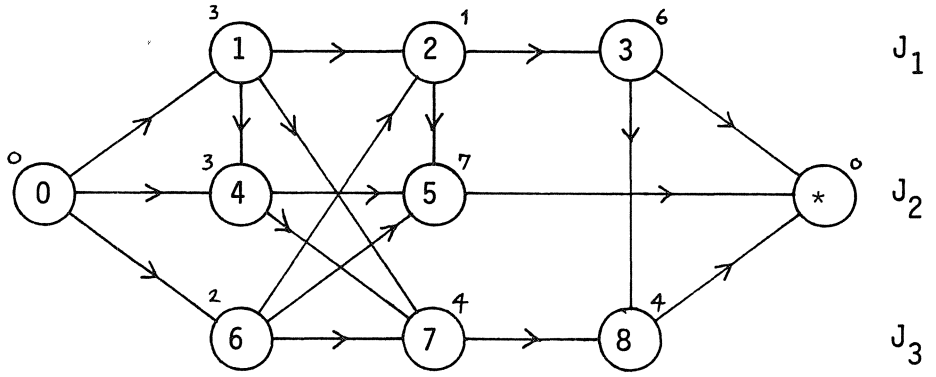

Figure 2 Example of an acyclic graph $G(D) = (V, C \cup D)$.

The general job-shop scheduling problem has been proved to be NP-complete [3], and several branch-and-bound algorithms have been developed in the past. In these algorithms, any node of the search tree corresponds to a partial solution which is characterized by a subset $D \subset \mathcal{D}$ of chosen arcs such that G(D) is acyclic.

Let us select an arbitrary M_k. For each O_r with $\mu(O_r) = M_k$ we can determine

- a *head*, i.e. the maximum weight of a path in G(D) from 0 to r;
- a *body*, i.e. the processing time p_r;
- a *tail*, i.e. the maximum weight of a path in G(D) from r to * minus p_r.

Furthermore, for O_r and O_s with $\mu(O_r) = \mu(O_s) = M_k$ we have a *precedence constraint* $O_r < O_s$ if G(D) contains a path from r to s.

It follows that on each machine we can define a one-machine problem of the type discussed in this paper and, since the arcs in $\mathcal{D}$-D are neglect-

ed, that the maximum of the m optimal solution values yields a lower bound on all extensions of D.

We conclude with some remarks on the combination of this lower bound and the two branching schemes that have been applied to the job-shop scheduling problem. A future paper will contain a more extensive discussion.

One branching scheme generates all active schedules [10], just as algorithm AS generates all active schedules for the one-machine problem (see section 3). The use of the above lower bound within such a scheme has led to the best job-shop scheduling algorithms developed so far [2;17] - although also non-active schedules may be generated by the branching scheme that is actually used [9] and some precedence constraints may be neglected in the one-machine problems.

In the second scheme branching takes place by adding either (r,s) or (s,r) to D [4;5]. The use of the above bound within this type of scheme requires a one-machine algorithm that is able to handle precedence constraints. Since the latter branching scheme seems very flexible in the sense that early branching decisions may settle essential conflicts within the problem, this approach merits serious consideration [15], the more so since the inclusion of precedence constraints turned out to influence the performance of the one-machine algorithms only to a very moderate extent.

7. A PRACTICAL APPLICATION

A practical scheduling situation in which the one-machine problem occurs arose in the context of the production of aluminium airplane parts. In a certain section of the factory in question, the production is centered around a rubber press. The metal pieces are first processed either by a *cutting* or by a *milling machine*. They next have to pass a *fitting shop* and subsequently have to spend a full working day in an *annealing furnace* before being pressed into their proper shape by the *rubber press*. After passing the fitting shop for a second time they are completely finished. The processing time of each operation is known in advance.

There are nine operators available to process the jobs. One of them operates the cutting and milling machines, six are working in the fitting

shop and two handle the rubber press; the annealing furnace requires no attention and can be assumed to have an infinite capacity, i.e., it can handle any number of jobs at the same time.

Since the rubber press is a relatively costly machine, the objective was to choose processing orders in such a way that the total completion time is minimized while idle time on the rubber press is avoided as much as possible.

If we denote the operations of job J_i by O_{ir} with processing times p_{ir} ($r = 1,\ldots,5$), typical data for a week's production look like those presented in the left-hand part of Table 5. Note that some jobs, which are left over from last week, have completed some of their initial operations.

We can model the above situation as a job-shop with four machines:

- M_1 represents the cutting and milling machines and has capacity 1;
- M_2 represents the fitting shop and has capacity 6;
- M_3 represents the annealing furnace and has capacity ∞;
- M_4 represents the rubber press and has capacity 1.

Each job has the same machine order (M_1,M_2,M_3,M_4,M_2).

Approaching the problem in a heuristic way, we note that

$$\sum_{i=1}^{35} p_{i1} = 56,\ \sum_{i=1}^{35} p_{i2} = 70,\ \sum_{i=1}^{35} p_{i4} = 48.5,\ \sum_{i=1}^{35} p_{i5} = 202.$$

Clearly, all jobs cannot be processed on M_1 and M_4 within one week of 40 hours and some overflow will result. It seems quite possible to schedule O_{i2} and O_{i3} directly after the completion of O_{i1}, but some waiting time for the jobs before the processing of O_{i4} and O_{i5} seems unavoidable. It is expedient to schedule O_{i1} in such a way that many jobs are quickly available for further processing, thereby taking p_{i4} and p_{i5} into account.

These intuitive considerations led to the following heuristic method, in which C_{ir} stands for the completion time of O_{ir}.

1. Schedule O_{i1} on M_1 according to nondecreasing $p_{i1}/(p_{i4}+p_{i5})$, thereby minimizing the weighted completion time $\sum_{i=1}^{35}(p_{i4}+p_{i5})C_{i1}$ (cf. [21]).
2. Schedule O_{i2} as early as possible on M_2 according to nondecreasing C_{i1}.
3. Schedule O_{i3} on M_3 according to $C_{i3} := 8\lceil C_{i2}/8\rceil+8$ ($\lceil x\rceil$ is the smallest integer not less than x).
4. Schedule O_{i4} on M_4 by solving the one-machine problem as discussed in this paper, defined by heads C_{i3}, bodies p_{i4} and tails p_{i5}.
5. Schedule O_{i5} as early as possible on M_2 according to nondecreasing C_{i4}.

A schedule resulting from application of this heuristic to the problem data is given by the completion times in Table 5; the corresponding Gantt-chart is shown in Figure 3. The one-machine problem on M_4 was solved by algorithm MF (see section 4); the first application of algorithm S yielded an optimal solution.

i	p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	$\frac{p_{i1}}{p_{i4}+p_{i5}}$	C_{i1}	C_{i2}	C_{i3}	C_{i4}	C_{i5}
1	4	4	8	2	6	.50	32	36	48	50	56
2	2	2	8	1	4	.40	22	24	32	33	38
3	1	1	8	1	2	.33	6	7	16	27.5	32
4	-	2	8	.5	6	-	0	2	16	22	32
5	6	2	8	2	7	.67	50	52	64	66	73
6	1	1	8	1	3	.25	4	5	16	24	33
7	1	1	8	.5	4	.22	2	3	16	22.5	31
8	-	5	8	.5	7	-	0	5	16	18.5	27
9	1	1	8	.5	4	.22	3	4	16	23	34
10	-	-	8	2	6	-	-	0	8	14	20
11	1	1	8	.5	2	.40	23	24	32	33.5	36
12	1	1	8	.5	2	.40	24	25	40	45	47
13	1	1	8	1	2	.33	7	8	16	28.5	34
14	2	1	8	1	4	.40	26	27	40	43	47
15	-	5	8	1	9	-	0	5	16	17	26
16	7	10	8	2	16	.39	20	30	40	42	58
17	-	-	-	1.5	6	-	-	-	0	6.5	12.5
18	1	1	8	1.5	6	.13	1	2	16	20	30
19	-	-	8	2	8	-	-	0	8	10	18
20	6	6	8	3	6	.67	56	62	72	75	81
21	-	-	-	2.5	12	-	-	-	0	2.5	15
22	-	-	8	2	7	-	-	0	8	12	19.5
23	-	-	-	2.5	11	-	-	-	0	5	16
24	2	2	8	1.5	3	.44	28	30	40	44.5	47.5
25	-	-	-	-	4	-	-	-	-	0	6
26	6	3	8	2.5	7	.63	44	47	56	61.5	68.5
27	-	-	8	1	5	-	-	0	8	15	20
28	1	1	8	1	3	.25	5	6	16	26.5	34
29	-	-	-	1.5	6	-	-	-	0	8	14
30	1	1	8	1	2	.33	8	9	24	29.5	35
31	1	1	8	1	2	.33	9	10	24	30.5	36
32	-	3	8	1	7	-	0	3	16	18	25
33	6	6	8	3	7	.60	38	44	56	59	66
34	-	6	8	1.5	6	-	0	6	16	21.5	31
35	4	2	8	1.5	10	.35	13	15	24	25.5	41

Table 5 A practical scheduling problem: data and results.

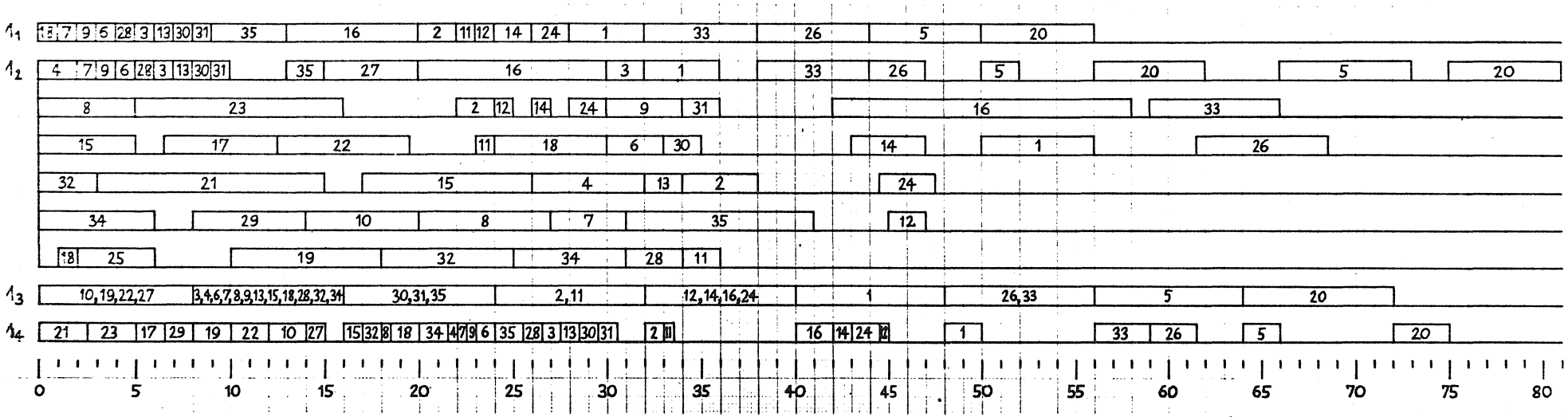

Figure 3 A practical scheduling problem: Gantt-chart.

The approach described above seems to be more generally applicable. Basically, it involves the determination of *critical machines* in the production process, i.e., the machines that are important from a cost minimizing point of view and on which the processing orders have a crucial influence on the quality of the schedule as a whole. The problem is then decomposed into problems involving one or more of those critical machines; these problems may be solved by methods inspired by sequencing theory. The resulting schedules are concatenated by suitable processing orders on the other machines leading to an overall schedule of reasonable quality.

Our experience with this heuristic approach has been limited to the small example above and our only conclusion would be that it seems to merit further experimentation. We feel that through this approach the models of machine scheduling theory, which may well correspond to an oversimplified picture of reality, may find application in varying situations that do not fit the standard models.

8. CONCLUDING REMARKS

The computational experience reported in section 5 leads us to conclude that the problem of minimizing maximum lateness on one machine can be satisfactorily solved by the algorithms described in sections 3 and 4. If solution by implicit enumeration is indeed unavoidable, there seems to be little room for further improvement.

It might be worth investigating if the ideas behind algorithms BS and MF could be applied to other machine scheduling problems. An interesting candidate is the problem of minimizing maximum completion time in a two-machine flow-shop with release dates. This problem can be interpreted as a variation on the three-machine model introduced in section 2: a non-bottleneck machine M_1 deals with the release dates and two bottleneck machines M_2 and M_3 constitute the flow-shop. Again, the case in which all r_i are equal can be solved in O(n log n) steps [13], whereas the general problem is NP-complete [3]. Similar remarks apply to the inverse problem, i.e. minimizing maximum lateness in a two-machine flow-shop.

The problem discussed in this paper finds application in theory and

practice, as has been demonstrated in sections 6 and 7. Especially the heuristic approach suggested in section 7 deserves further examination. It might be a suitable response to the frequent complaint about the lack of successful practical applications of machine scheduling theory.

ACKNOWLEDGEMENTS

We are grateful to M. Florian and G.B. McMahon for valuable discussions concerning their algorithm. Also, we would like to thank J.H. Galjaard and J.S. Knipscheer of the Graduate School of Management, Delft, for providing the data for the practical scheduling problem.

REFERENCES

1. K.R. BAKER, Z.-S. SU (1974) Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Res. Logist. Quart.* 21,171-176.
2. P. BRATLEY, M. FLORIAN, P. ROBILLARD (1973) On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{max})$ problem. *Naval Res. Logist. Quart.* 20,57-67.
3. P. BRUCKER, J.K. LENSTRA, A.H.G. RINNOOY KAN (1975) Complexity of machine scheduling problems. Report BW 43, Mathematisch Centrum, Amsterdam.
4. J.M. CHARLTON, C.C. DEATH (1970) A method of solution for general machine scheduling problems. *Operations Res.* 18,689-707.
5. J.M. CHARLTON, C.C. DEATH (1971) A generalized machine scheduling algorithm. *Operational Res. Quart.* 21,127-134.
6. R.W. CONWAY, W.L. MAXWELL, L.W. MILLER (1967) *Theory of Scheduling*. Addison-Wesley, Reading, Mass.
7. M.I. DESSOUKY, C.R. MARGENTHALER (1972) The one-machine sequencing problem with early starts and due dates. *AIIE Trans.* 4,214-222.
8. J. EDMONDS (1965) Paths, trees, and flowers. *Canad. J. Math.* 17,449-467.
9. M. FLORIAN, P. TREPANT, G. McMAHON (1971) An implicit enumeration algorithm for the machine sequencing problem. *Management Sci.* 17,B782-792.

10. B. GIFFLER, G.L. THOMPSON (1960) Algorithms for solving production scheduling problems. *Operations Res.* 8,487-503.

11. W.A. HORN (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21,177-185.

12. J.R. JACKSON (1955) Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

13. S.M. JOHNSON (1954) Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Quart.* 1,61-68.

14. E.L. LAWLER (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.* 19,544-546.

15. J.K. LENSTRA, A.H.G. RINNOOY KAN (1973) Towards a better algorithm for the job-shop scheduling problem - I. Report BN 22, Mathematisch Centrum, Amsterdam.

16. J.K. LENSTRA, A.H.G. RINNOOY KAN (1975) A recursive approach to the generation of combinatorial configurations. Report BW 50, Mathematisch Centrum, Amsterdam.

17. G. McMAHON, M. FLORIAN (1975) On scheduling with ready times and due dates to minimize maximum lateness. *Operations Res.* 23,475-482.

18. A.H.G. RINNOOY KAN (1973) The machine scheduling problem. Report BW 27, Mathematisch Centrum, Amsterdam.

19. B. ROY, B. SUSSMANN (1964) Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS no.9 bis, SEMA, Montrouge.

20. L. SCHRAGE (1971) Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript.

21. W.E. SMITH (1956) Various optimizers for single-state production. *Naval Res. Logist. Quart.* 3,59-66.