stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 33/74 APRIL

IN SAMENWERKING MET HET
INTERUNIVERSITAIR INSTITUUT BEDRIJFSKUNDE
DELFT

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

MINIMIZING TOTAL COSTS IN ONE-MACHINE SCHEDULING

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 TOTAL COSTS IN ONE-MACHINE SCHEDULING

A.H.G. RINNOOY KAN
Interfaculty for Graduate Studies in Management, Delft, The Netherlands

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

ABSTRACT

We consider the following problem. Suppose n jobs have to be processed on a machine which can handle only one job at a time. For each job we have a fixed processing time and a cost function which is non-decreasing in its finishing time. We want to find a schedule that minimizes total costs.

After reviewing the relevant work done so far on this problem, we present a new algorithm for a general cost function. The algorithm is tested for the well known case of a weighted tardiness cost function.

NOTE

This report is not for review. It has been submitted for publication in a journal.

CONTENTS

0. INTRODUCTION

We consider the following problem. Suppose we have n *jobs* $J_1,\ldots,J_n$, to be processed on a *machine* which can handle only one job at a time. Each job J_i $(i = 1,\ldots,n)$ takes p_i time-units to be processed. With each job J_i $(i = 1,\ldots,n)$ is associated a non-decreasing *cost function* $c_i(t)$, representing the cost incurred if J_i finishes at time t. Any processing schedule leads to a finishing time t_i for J_i $(i = 1,\ldots,n)$. We want to find a schedule that minimizes $\sum_{i=1}^{i=n} c_i(t_i)$.

Various special cases corresponding to specific cost functions have been studied in the past. Below, we shall briefly review the relevant work done so far, before introducing a new and completely general branch-and-bound algorithm to solve the problem described above. To test the computational efficiency of the algorithm, we consider the well known case of a weighted tardiness cost function and compare the algorithm's performance to that of an algorithm specifically designed for this cost function.

1. PREVIOUS WORK

The above-mentioned problem has been solved efficiently only for the case of a linear cost function

$$c_i(t) = \alpha_i(t - d_i). \qquad (1)$$

Interpreting d_i as a *due-date* for J_i $(i = 1,\ldots,n)$, we see that (1) represents the *weighted lateness* criterium. Putting $d_i = 0$ for all i gives the *weighted flowtime* criterium; also included are the special cases of *average lateness* and *average flowtime* where $\alpha_i = 1/n$ for all i. Smith [20] proved that all these problems are solved by processing the jobs in order of increasing p_i/α_i ratio.

Apart from Schild's and Fredman's work [18] on a quadratic cost function and dynamic programming formulations for the general problem by Held and Karp [12] and Lawler [14], most researchers have concentrated on the *weighted tardiness* cost function

$$c_i(t) = \alpha_i \max\{0,t - d_i\} \qquad (2)$$

which includes the *average tardiness* criterium as a special case with $\alpha_i = 1/n$ for all i. The attractiveness of (2) as a cost function is due to the fact that is seems economically realistic to introduce cost functions $c_i(t)$ that become effective only if J_i finishes after its due-date d_i. Also, (2) looks relatively simple from a computational point of view.

However, finding an optimal schedule with respect to (2) has proved to be a very difficult task. The problem is included in Karp's list [13] of problems that are expected not to be solvable within an amount of time that is polynomial in the number of input parameters. This indicates that an efficient general solution method is not likely to be found. It is therefore not surprising that all methods so far seek to find the optimal schedule by implicitly enumerating the large, but finite set of feasible schedules. Two main approaches can be distinguished.

The first one relies heavily on so-called *elimination criteria*. Application of these criteria may show that a certain job necessarily precedes or follows one or more other jobs in at least one optimal schedule. By

determining as many of these precedence relations as possible, a number of feasible schedules can be eliminated. The collection of remaining schedules, containing at least one optimal one, is either explicitly enumerated or implicitly enumerated by dynamic programming or simple branch-and-bound methods. The best example of this first approach is given by the work of Emmons [8] on the average tardiness problem. The fact that all weights are equal facilitates the development of powerful elimination criteria. Emmons' criteria were successfully implemented by Srinivasan [21], who used dynamic programming to enumerate the schedules that could not be eliminated.

For the weighted tardiness cost function and *a fortiori* for the completely general cost function, no really powerful elimination criteria have been found so far. The other approach has therefore been to rely in the first place on efficient implicit enumeration methods of the *branch-and-bound* type and to use at most a few simple elimination criteria. A good example of this approach is Shwimer's work [19] on the weighted tardiness problem. We shall return to Shwimer's algorithm below.

In general, the performance of branch-and-bound algorithms has been rather disappointing. Baker and Martin [1], who compared a rumber of average tardiness algorithms, concluded that this was due to the fact that the lower bounds developed until then were not very satisfactory. Only one or two jobs out of a subset of jobs on the processing costs of which a lower bound was sought, actually contributed to this lower bound. The lower bound presented below at least does not suffer from this defect.

The present state-of-the-art in both average and weighted tardiness scheduling is such that problems with 12 or 15 jobs may already present severe computational problems. No algorithm for a general cost function has been tested on a large scale so far.

2. A NEW ALGORITHM

In what follows we shall present a new branch-and-bound algorithm for a general cost function. In section 2.1 we outline the *enumeration scheme*, sections 2.2 and 2.3 introduce some *elimination criteria* and remarks on their *implementation*, and in sections 2.4. and 2.5 the *lower bound* and its *implementation* are discussed.

For the sake of an easy notation, we define

$$P(Q) = \sum_{i \in Q} p_i$$

for any $Q \subset \{1,\ldots,n\}$.

2.1. Enumeration scheme

The enumeration scheme that we shall use fills a schedule from back to front (cf. [14]). This is possible because it can be proved [4,17] that there always exists an optimal schedule without gaps (i.e. machine idle time). The total time needed to process a set of jobs can therefore be determined in advance and is independent of the processing order.

We create a search tree as follows. From the root node, where no jobs have been scheduled, we branch to n different nodes on the first level, each node corresponding to a specific job being scheduled in the n-th position. Each of these nodes leads to n-1 new nodes on the second level, corresponding to one of the remaining n-1 jobs filling the (n-1)-th position.

More generally, each node is characterized by a set $\{J_i\}_{i \in S'}$ (with $S' \subset \{1,\ldots,n\}$) of jobs, that in a given order fill the $(n - |S'| + 1)$-th to the n-th position in the schedule. Denoting $\{1,\ldots,n\} - S'$ by S, $|S|$ new branches are created by successively placing each job J_r $(r \in S)$ in the $|S|$-th position of the schedule. This job then runs from $P(S) - p_r$ to $P(S)$.

The n! nodes on the n-th level of the search tree represent all the possible processing schedules.

2.2. Elimination criteria

In each node, characterized by a set $\{J_i\}_{i \in S}$ of unscheduled jobs, we try to apply elimination criteria, based on the theorems in this section. Throughout, our theorems hold for general non-decreasing cost functions; the implications for the special case of weighted tardiness functions are formulated as corollaries.

Suppose that we have found that there exists an optimal schedule whereby, for $i \in S$, the set $\{J_h | h \in B_i\}$ precedes J_i and the set $\{J_h | h \in A_i\}$ follows J_i. Any established relation "J_j precedes J_k" implies that $j \in B_k$ and $k \in A_j$. In the following, we restrict ourselves to schedules which satisfy the precedence constraints, defined by B_i and A_i ($i \in S$).

THEOREM 1. *If for two jobs* J_j *and* J_k ($j,k \in S$) *we have*

(a) $c_j(t) - c_k(t)$ *is a non-decreasing function of* t *on the interval* $(P(B_k) + p_k, P(S-A_j))$, *and*

(b) $p_j \leq p_k$,

then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. Consider any schedule whereby J_k precedes J_j. Denote by C the starting time of J_k and by D the finishing time of J_j. Compare this schedule with the schedule obtained by interchanging J_k and J_j (*see* Figure 1).

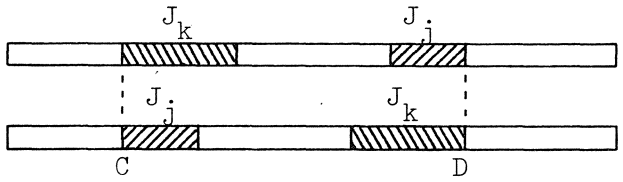

Figure 1

The contribution to total costs by all jobs except J_k decreases or remains the same, as can be easily checked. As to J_k, it follows from

$$P(B_k) + p_k \leq C + p_k \leq D \leq P(S-A_j) \tag{3}$$

and condition (a) that

$$c_j(D) - c_k(D) \geq c_j(C+p_k) - c_k(C+p_k). \tag{4}$$

Because of condition (b), we have

$$c_j(C+p_k) \geq c_j(C+p_j). \tag{5}$$

Together, (4) and (5) imply

$$c_j(D) + c_k(C+p_k) \geq c_j(C+p_j) + c_k(D),$$

which means that the joint contribution of J_j and J_k to total costs also decreases or remains the same. (Q.E.D.)

Remark. If both $c_j(t)$ and $c_k(t)$ are continuous and almost everywhere differentiable on $(P(B_k) + p_k, P(S-A_j))$, then condition (a) is trivially fulfilled if $dc_j(t)/dt \geq dc_k(t)/dt$ for every $t \in (P(B_k) + p_k, P(S-A_j))$ where both derivatives are defined.

COROLLARY 1.1. *If* $c_i(t) = \alpha_i \max\{0, t - d_i\}$ *for all* i, *and if for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*

(a) $d_j \leq d_k$,

(b) $\alpha_j \geq \alpha_k$, *and*

(c) $p_j \leq p_k$,

then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. We can apply Theorem 1 with $B_k = A_j = \emptyset$. Clearly, conditions (a) and (b) imply that $c_j(t) - c_k(t)$ is non-decreasing on the interval $(0, P(S))$ (*see* Figure 2). (Q.E.D.)

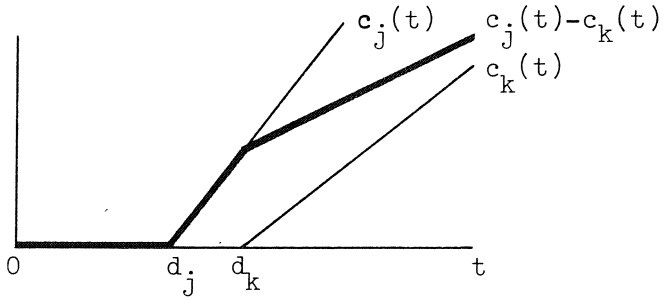

Figure 2

COROLLARY 1.2. *If* $c_i(t) = \alpha_i \max\{0, t - d_i\}$ *for all* i, *and if for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*

(a) $d_j \leq P(B_k) + p_k$,

(b) $\alpha_j \geq \alpha_k$, *and*

(c) $p_j \leq p_k$,

then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. Condition (a) implies that $c_j(t) > 0$ for $t > P(B_k) + p_k$, and it follows from condition (b) that $c_j(t) - c_k(t)$ is non-decreasing on the required interval. (Q.E.D.)

THEOREM 2. *If for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*
(a) $c_k(P(B_k) + p_k) = c_k(P(S-A_j) - p_k)$, *and*
(b) $c_j(t) - c_k(t)$ *is a non-decreasing function of* t *on the interval* $(P(S-A_j) - p_k, P(S-A_j))$,
then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. Clearly, conditions (a) and (b) imply that $c_j(t) - c_k(t)$ is non-decreasing on the interval $(P(B_k) + p_k, P(S-A_j))$, so in the case that $p_j \leq p_k$ we can apply Theorem 1. Suppose now that $p_j > p_k$. Again, consider any schedule whereby J_k precedes J_j, denote by C the starting time of J_k and by D the finishing time of J_j. Compare this schedule with the schedule obtained by putting J_k directly after J_j (*see* Figure 3). The contribution to total costs

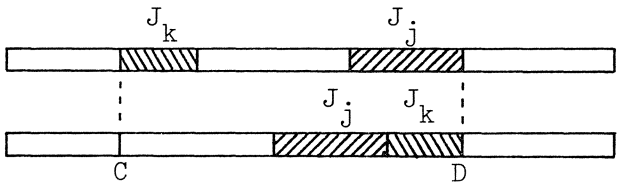

Figure 3

by all jobs except J_k decreases or remains the same. As to J_k, it follows from

$$P(B_k) + p_k \leq C + p_k < D - p_k \leq P(S-A_j) - p_k$$

and condition (a) that

$$c_k(D-p_k) = c_k(C+p_k). \tag{6}$$

Because of condition (b), we have

$$c_j(D) - c_k(D) \geq c_j(D-p_k) - c_k(D-p_k). \tag{7}$$

Together, (6) and (7) imply

$$c_j(D) + c_k(C+p_k) \geq c_j(D-p_k) + c_k(D),$$

which means that the joint contribution of J_j and J_k to total costs also decreases or remains the same. (Q.E.D.)

COROLLARY 2.1. *If* $c_i(t) = \alpha_i \max\{0,t - d_i\}$ *for all* i, *and if for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*

(a) $d_k \geq P(S-A_j) - p_k$,

(b) $d_j \leq d_k$, *and*

(c) $\alpha_j \geq \alpha_k$,

then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. Condition (a) implies that $c_k(P(B_k) + p_k) = c_k(P(S-A_j) - p_k)$, and it follows from conditions (b) and (c) that $c_j(t) - c_k(t)$ is non-decreasing on the interval $(0,P(S))$ (*see* Figure 2). (Q.E.D.)

THEOREM 3. *If for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*

$$c_k(P(B_k) + p_k) = c_k(P(S-A_j)), \tag{8}$$

then we only have to consider schedules whereby J_j *precedes* J_k.

Proof. We can apply Theorem 2, since its conditions (a) and (b) follow from (8). (Q.E.D.)

COROLLARY 3.1. *If* $c_i(t) = \alpha_i \max\{0,t - d_i\}$ *for all* i, *and if for two jobs* J_j *and* J_k $(j,k \in S)$ *we have*

$$d_k \geq P(S-A_j),$$

then we only have to consider schedules whereby J_j *precedes* J_k.

THEOREM 4. *If for a job* J_k $(k \in S)$ *we have*

$$c_k(p_k) = c_k(P(S)), \tag{9}$$

then we only have to consider schedules whereby J_k *comes last among the jobs in* S.

Proof. Condition (9) implies (8) for all $j \in S$, so we can apply Theorem 3 to each pair (J_j,J_k) with $j \in S - \{k\}$. (Q.E.D.)

COROLLARY 4.1. *If* $c_i(t) = \alpha_i \max\{0,t - d_i\}$ *for all* i, *and if for a job* J_k $(k \in S)$ *we have*

$$d_k \geq P(S),$$

then we only have to consider schedules whereby J_k *comes last among the jobs in* S.

Corollary 1.1 can be found in Shwimer [19]. Corollaries 1.2 and 2.1 are extended versions of Theorems 1 and 2 of Emmons [8]. Our proofs, however, are considerably simpler than the original ones. Corollary 4.1 is also known as *Elmaghraby's Lemma* [7].

2.3. Implementation of the elimination criteria

The main problem arising with the implementation of a number of elimination criteria is the possible creation of *precedence cycles*: it is perfectly imaginable that two theorems contradict each other. The nature of our theorems and corollaries, however, is such that applying them successively, while guarding against precedence cycles, will always lead to a collection of schedules containing at least one optimal one. We avoid the creation of precedence cycles by immediately constructing the *transitive closure* of each precedence relation "J_j precedes J_k" that we find: then $A_i := A_i \cup \{k\} \cup A_k$ for every $i \in \{j\} \cup B_j$ and $B_\ell := B_\ell \cup \{j\} \cup B_j$ for every $\ell \in \{k\} \cup A_k$. If we furthermore restrain ourselves to examining pairs (ℓ,i) between which no relation has been found so far, we can never find a precedence cycle. For if we found that "J_ℓ precedes J_i" and if it then turned out that $j \in B_k$ for some $j \in A_i$, $k \in B_\ell$, then we would have set $\ell \in A_i$, $i \in B_\ell$ in a previous stage and therefore would not have examined this pair anew.

In the case of general cost functions we can apply Theorems 1 to 4 in every node: the set S decreases and the sets A_i and B_i increase in size as we progress through the search tree. In the case of weighted tardiness functions we apply Corollary 4.1 in every node, whereas Corollaries 1.1, 1.2, 2.1 and 3.1 are only used in the root node with $S = \{1,\ldots,n\}$. Theoretically, Corollaries 1.2, 2.1 and 3.1 could be rechecked in every node, but the advantages of doing so are in this case outweighed by the disadvantages of complicated and time-consuming bookkeeping.

Corollaries 1.1, 1.2, 3.1 and 2.1 are now implemented by running through them in this order, while keeping in mind the above remarks, and repeating this process until no further improvements are possible. If after this process the earliest possible finishing time $P(B_k) + p_k$ of a job J_k is larger than its due-date d_k, then this due-date can be set equal to $P(B_k) + p_k$, thereby incurring costs $\alpha_k(P(B_k) + p_k - d_k)$ and increasing the chances on application of Corollary 4.1.

The latter corollary is checked in every node. To avoid contradictions with the precedence relations, found previously by the other corollaries, a total ordering has to be constructed from this partial ordering. Each pair of jobs (J_j,J_k) is ordered by a precedence relation from the partial order-

ing, or else by increasing $\min\{h|h \in \{i\} \cup A_i\}$ $(i = j,k)$. On entering a node, we then check if the last unscheduled job in this total ordering satisfies the condition mentioned in Corollary 4.1.

It seems useful to point out that precedence relations that are *a priori* given can now be dealt with in an obvious way, broadening the scope of possible applications of our algorithm.

2.4. Lower bound

Suppose again that a subset $\{J_i\}_{i\in S'}$ of jobs fills the $(n - |S'| + 1)$-th to the n-th position in the schedule and that the subset $\{J_i\}_{i\in S}$ with $S = \{1,\ldots,n\} - S'$ remains to be scheduled. A lower bound LB on the costs of all possible schedules in this node will have the form

$$LB = c(S') + LB^*, \tag{10}$$

where $c(S')$ denotes the known total costs incurred by the jobs in $\{J_i\}_{i\in S'}$ and LB^* is a lower bound on the total costs of scheduling the jobs in $\{J_i\}_{i\in S}$ in the period from 0 to $P(S)$.

To compute LB^*, we put $m = |S|$ and renumber the jobs in $\{J_i\}_{i\in S}$ from 1 up to m. Our lower bound is now based on the observation that, if

$$p_1 = p_2 = \ldots = p_m = p,$$

an optimal schedule for $\{J_1,\ldots,J_m\}$ can easily be found as follows (cf. [14]). Defining

$$c_{ij} = c_i(jp),$$

an optimal schedule π^* with minimal total costs is given by the solution to the following *linear assignment problem*:

$$\min_\pi \sum_{j=1}^{j=m} c_{\pi(j)j}, \tag{11}$$

where $\pi = (\pi(1),\ldots,\pi(m))$ runs over all permutations of $\{1,\ldots,m\}$.

If not all p_i $(i = 1,\ldots,m)$ are equal, an optimal schedule cannot be obtained so easily, but the above idea can now be used to compute a lower bound. This can be done in two ways.

Assuming all p_i are integers, we can find their greatest common divisor g and treat each job J_i as a sequence of p_i/g new jobs of equal length g. The problem, originally given by (11), now becomes a $(P(S)/g) \times m$ *linear transportation problem*, that produces a lower bound if we succeed in defining appropriate cost coefficients c_{ij}. For the case that

$$c_i(t) = \alpha_i \max\{0, t - d_i\} + \beta_i t,$$

Gelders en Kleindorfer [10,11] have developed suitable cost coefficients. Three problems remain with this approach.

(1) Generally, the optimal solution to the transportation problem involves job splitting (preemption), giving an infeasible schedule. Provisions must be made for this.

(2) As the greatest common divisor g of all p_i will usually be equal to 1, the size of the transportation problem tends to be very large. It then becomes practically impossible to solve this problem in every node again.

(3) It seems to be difficult to define effective cost coefficients for a general cost function $c_i(t)$.

For the above reasons, we prefer a different approach, which basically boils down to redefining the cost coefficients for the m×m assignment problem, so that c_{ij} becomes a lower bound on the cost of putting J_i in the j-th position in the schedule. To accomplish this, we compute the earliest possible finishing time t_{ij} of job J_i if we put J_i in the j-th position in the schedule and if we observe the precedence constraints, given by B_i and A_i. Using the notation

$$R_i(k) = \min_Q P(Q \mid Q \subset \{1,\ldots,m\} - (B_i \cup \{i\} \cup A_i), |Q| = k),$$

t_{ij} is given by

$$t_{ij} = P(B_i) + p_i + R_i(j - |B_i| - 1)$$

for $|B_i| < j \leq |\{1,\ldots,m\} - A_i|$, as can be easily checked. Redefining

$$c_{ij} = \begin{cases} c_i(t_{ij}) & \text{for } |B_i| < j \leq |\{1,\ldots,m\} - A_i| \\ \infty & \text{otherwise,} \end{cases}$$

and using these cost coefficients in problem (11) now gives the desired lower bound LB^*. This is easily proved as follows. If an optimal schedule

for our original problem is given by a permutation π^* with minimal costs $c(\{1,\ldots,m\})$, then certainly

$$\sum_{j=1}^{j=m} c_{\pi^*(j)j} \leq c(\{1,\ldots,m\}) \tag{12}$$

because of the definition of c_{ij} and the fact that $c_i(t)$ is non-decreasing. We also have

$$LB^* \leq \sum_{j=1}^{j=m} c_{\pi^*(j)j} \tag{13}$$

because π^* is a feasible solution to problem (11). Together, (12) and (13) imply the validity of the lower bound LB^*.

2.5. Implementation of the lower bound

Suppose we are now in the following situation. In the current node a lower bound LB has been computed, and the jobs in the set $\{J_i \mid i \in S, S \cap A_i = \emptyset\}$ are candidates for the m-th position in the schedule. Choosing any of them leads to a new node in the search tree. Fortunately, we can do better than solving *ab initio* the assignment problem in each of these *descendant* nodes, by exploiting the solution to the assignment problem in the current *parent* node. This problem is given by (11) and can be reformulated as

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{i=m} \sum_{j=1}^{j=m} c_{ij}\, x_{ij} \\ &\text{subject to} && \sum_{j=1}^{j=m} x_{ij} = 1 \quad \text{for} \quad i = 1,\ldots,m, \\ &&& \sum_{i=1}^{i=m} x_{ij} = 1 \quad \text{for} \quad j = 1,\ldots,m, \\ &&& x_{ij} \geq 0 \quad \text{for} \quad i,j = 1,\ldots,m. \end{aligned} \tag{14}$$

Its dual problem is given by

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{i=m} u_i + \sum_{j=1}^{j=m} v_j \\ &\text{subject to} && u_i + v_j \leq c_{ij} \quad \text{for} \quad i,j = 1,\ldots,m. \end{aligned}$$

An optimal solution to these problems has the value LB^* and is denoted by

(x^*_{ij}) and (u^*_i, v^*_j), respectively. The value of the best schedule found so far is denoted by UB and gives an upper bound on the costs of an optimal schedule.

In the parent node, we can with little computational effort obtain a lower bound LB_r on the costs of scheduling job J_r in the m-th position. The costs of the scheduled jobs are equal to

$$c(S' \cup \{r\}) = c(S') + c_{rm}. \tag{15}$$

The costs of the remaining jobs can be bounded from below by the solution to the assignment problem which arises from (14) by removing the r-th row and the m-th column. Since $(u^*_i, v^*_j)_{i \neq r, j \neq m}$ is a feasible dual solution to this problem, the optimal solution is not smaller than

$$\sum_{i \neq r} u^*_i + \sum_{j \neq m} v^*_j = LB^* - u^*_r - v^*_m. \tag{16}$$

Combining (10), (15) and (16), we define

$$LB_r = LB + c_{rm} - u^*_r - v^*_m \geq LB.$$

Clearly, any potential descendant node for which $LB_r \geq UB$ can be eliminated.

From the remaining candidate jobs a job J_r with minimal LB_r is scheduled in the m-th position, and we start to explore the corresponding descendant node. Application of the elimination criteria in this node may increase LB_r. For example, if in the case of a weighted tardiness cost function a job J_k is scheduled in position m-1 by application of Corollary 4.1, then we have $c_{k\,m-1} = 0$ and LB_r can be replaced by

$$LB_r - u^*_k - v^*_{m-1} \geq LB_r.$$

However, if this new LB_r does not lead to elimination of the node, we have to solve its assignment problem. Indexing the jobs as in the parent node, and considering only indices which correspond to unscheduled jobs or unfilled positions, we can still profit from the optimal solution to the assignment problem in the parent node in the following ways.

(1) (x^*_{ij}) is a partial (i.e. possibly non-feasible) solution to the new assignment problem and can serve as a starting point for finding an optimal solution.

(2) It is easily seen that, if we pass from the parent into the descendant node, the earliest finishing times t_{ij} will not decrease, neither will the cost coefficients c_{ij}. So (u_i^*, v_j^*) is a feasible dual solution to the new assignment problem.

(3) If our assignment algorithm requires that these initial primal and dual solutions are orthogonal, i.e.

$$u_i^* + v_j^* = c_{ij} \quad \text{if} \quad x_{ij}^* > 0,$$

then this can be achieved by resetting $x_{ij}^* = 0$ only if

$$u_i^* + v_j^* < c_{ij} \quad \text{and} \quad x_{ij}^* = 1.$$

Solving the new assignment problem may lead to elimination in two ways.

(1) If our assignment algorithm yields a sequence of non-decreasing feasible dual solutions (u_i, v_j), then we can eliminate the node as soon as $c(S') + \Sigma u_i + \Sigma v_j \geq UB$.

(2) In general, we can eliminate the node if the value of the optimal solution to the new assignment problem is not smaller than UB.

The above discussion suggests an alternative bounding mechanism. By solving an assignment problem only in the root node, we can compute lower bounds throughout the whole search tree by means of sums of appropriate dual variables. In fact, this idea has been implemented by Gelders and Kleindorfer [10,11]: it is simply not feasible to find a new optimal solution to their large transportation problem in every node again. However, we rejected this approach, and our computational results point out that even our sharper bounds may lead to quite large search trees for problems of a moderate size.

The question now arises which algorithm has to be used to solve the assignment problems. Such an algorithm should be fast, and should yield a dual solution in order to prune the search tree at an early stage. Preferably, it should be able to use any initial solution on which no special requirements are imposed, such as being orthogonal or, even worse, being basic. Finally, it ought to supply a sequence of non-decreasing feasible dual solutions, so that we can eliminate the current node without having solved the assignment problem completely. Among known primal

algorithms, Beale's algorithm [2] and the well known stepping-stone algorithm [3] deserve serious consideration. The Hungarian method [9] is a representative example of a mixed primal-dual algorithm, whereas Dorhout's algorithm [5] represents a dual approach to the problem. Table I indicates the quality of these four algorithms with respect to the desirable properties stated above.

TABLE I

PROPERTIES OF ASSIGNMENT ALGORITHMS

Property	Algorithm			
	Stepping-Stone	Beale	Hungarian	Dorhout
fastness (cf.[5])	o	o	-	+
dual solution	+	-	+	+
no requirements on initial solution	-	+	o	o
increasing feasible dual solutions	-	-	+	+

After some experiments with Beale's method which has the serious drawback of not producing dual solutions, we settled for Dorhout's algorithm, which can be considered as a synthesis of ideas proposed by Tabourier [22] and Tomizawa [23]. Essentially, the algorithm works on a complete *bipartite graph* $G = (S,M,\Gamma)$, where the vertex sets S and M correspond to the sets of unscheduled jobs and unfilled positions in the current node, respectively; each edge $e_{ij} \in \Gamma$ $(i \in S, j \in M)$ has a weight $w_{ij} = c_{ij} - u_i - v_j$. The algorithm starts with a feasible dual solution (u_i, v_j) and a partial primal solution (x_{ij}), orthogonal to the dual one. The partial solution (x_{ij}) defines a *matching* on G. The algorithm takes any *exposed* vertex $i \in S$ as starting point and constructs the shortest augmenting path from i to the nearest exposed vertex in M, using Dijkstra's shortest path algorithm [6]. Having found such a path, it augments the matching, and changes the dual solution in such a way that its feasibility is maintained and the orthogonality is restored.

3. COMPUTATIONAL EXPERIMENTS

3.1. Tested algorithms

As announced before, we tested our general method on the well known case of a weighted tardiness cost function.

As a comparable alternative approach, we also tested Shwimer's algorithm [19] for this cost function. His enumeration scheme is equivalent to ours. He applies only two elimination criteria, formulated here as Corollaries 4.1. and 1.1. His lower bound tries to eliminate potential descendants in the parent node; instead of LB_r he uses

$$LB'_r = c(S' \cup \{r\}) + \\ + \min_{i \in S-\{r\}} \left\{ \alpha_i \max\{0, P(S-\{r\}) - d_i\} + \min_{h \in S-\{r,i\}} \{\alpha_h\} \cdot T_{max}(S-\{r,i\}) \right\},$$

where $T_{max}(Q)$ denotes the minimal maximal tardiness over all possible schedules of the set $\{J_h\}_{h \in Q}$, found by ordering these jobs according to increasing due-dates (*see* [4,17]). It is clear that Shwimer's bound can be computed much quicker than our bound LB, but that only two jobs really contribute to its value. Moreover, Shwimer's bound depends explicitly on a property of the tardiness function. It is possible to find the minimal maximal cost incurred by a set of m-1 jobs for a general cost function due to a device of Lawler [15], but the number of operations then increases from $O(m \log_2 m)$ to $O(m^2)$.

As a third, cruder approach, a lexicographic enumeration algorithm (cf. [16]) was tested. This method enumerates schedules in the way described in section 2.1, always choosing J_r such that d_r is maximal over all remaining candidates in $\{J_i\}_{i \in S}$. Corollary 4.1 can then easily be applied; no other elimination criteria have been incorporated. Also a simple bounding mechanism is used, with

$$LB''_r = c(S' \cup \{r\}).$$

For a more general remark on the possible use of such a quick complete enumeration method, we refer to section 4.

3.2. Testproblems

We shall now describe in detail the way in which we generated random data on which to test these three methods. The reasons for this detailed approach will become apparent as we proceed.

Given the number of jobs n, each problem is completely specified by 3n values (p_i, d_i, α_i), i = 1,...,n. These values correspond to processing times, due-dates and weights, respectively. We regard the n triples as a three-dimensional sample from a joint distribution with density function f(x,y,z).

In all our tests, the third random variable $\underline{\alpha}$ is independent of $\underline{p}$ and $\underline{d}$. We have

$$f(x,y,z) = f_{pd}(x,y)\ f_{\alpha}(z),$$

where $\underline{\alpha}$ is uniformly distributed over the interval (4.5, 15.5).

In what follows, we shall introduce four parameters that determine $f_{pd}(x,y)$ and that we believed *a priori* to be of possible influence on any algorithm's performance. In fact, three of them are already mentioned as such by Srinivasan [21] and Baker and Martin [1]. Their work indicates that the choice of a particular function may have a strong influence on the performance of any tardiness algorithm in a way that may be characteristic for the algorithm in question.

The first parameter measures the *correlation between* $\underline{p}$ *and* $\underline{d}$, $\rho(\underline{p},\underline{d})$. It is intuitively plausible that there may be a significant difference between problems where longer jobs tend also to have later due-dates, and problems where there is no correlation whatsoever. If $\alpha_i = 1/n$ for all i, then a problem with perfect correlation can be trivially solved by ordering the jobs according to non-decreasing d_i (*see* Emmons [8]). To investigate the influence of correlation we use two different sorts of functions $f_{pd}(x,y)$. Either

$$f_{pd}(x,y) = f_p(x)\ f_d(y),$$

in which case $\underline{p}$ and $\underline{d}$ are independent random variables and $\rho(\underline{p},\underline{d}) = 0$, or

$$f_{pd}(x,y) = f_p(x)\ f_{d|p}(y|x),$$

in which case the due-date generated depends explicitly on the processing time and $\rho(\underline{p},\underline{d})$ depends on the particular form of the density functions involved.

In both cases, $\underline{p}$ is normally distributed with expectation μ_p and variance σ_p^2. We arbitrarily fix $\mu_p = 100$. With regards to σ_p^2 however, we have to introduce as the second possibly significant parameter the *relative variation of processing times* $s = \sigma_p/\mu_p$. We introduce s because our lower bound will presumably be sharper when processing times differ relatively little, as will be obvious from section 2.3. Hence, we may expect problems with small s to be relatively easy for our algorithm.

In the case of non-correlated $\underline{p}$ and $\underline{d}$, $\underline{d}$ is uniformly distributed with expectation μ_d and variance $\sigma_d^2 = \lambda_d^2/12$, where λ_d denotes the length of the interval on which $f_d(y) > 0$.

We fix μ_d by introducing as a third parameter the *average tardiness factor* $t = 1 - \mu_d/(n\mu_p)$. The value of t roughly indicates the average fraction of jobs that will be late (cf.[1]). Problems with $t = 1$ or $t = 0$ tend to be easy: if all jobs are late, then ordering the jobs according to non-decreasing p_i/α_i produces an optimal schedule, and if we find by ordering the jobs according to non-decreasing d_i that no job is late, then clearly this schedule is optimal (Emmons [8] gives slightly stronger versions of these theorems). Srinivasan [21] found problems where t is near 0.65 to be most difficult.

Finally, λ_d is fixed by the fourth parameter, the *relative range of due-dates* $r = \lambda_d/(n\mu_p)$. Intuitively, a large r increases the number of times that Corollaries 1.1 and 2.1 can be applied, thereby speeding up computations.

In the case of correlated $\underline{p}$ and $\underline{d}$, $\underline{d}|\underline{p}=p$ is again uniformly distributed, with $\mu_{d|p}$ and $\lambda_{d|p}$ specified analogously by $t = 1 - \mu_{d|p}/(np)$ and $r = \lambda_{d|p}/(np)$. Specific values of s, t and r determine the value of $\rho(\underline{p},\underline{d})$. We have

$$\rho(\underline{p},\underline{d}) = \frac{1 - t}{\sqrt{(1+1/s^2)r^2/12 + (1-t)^2}},$$

as can be established by straightforward calculations.

Choosing for non-correlated or correlated $\underline{p}$ and $\underline{d}$, and fixing s, t and

r, we can generate n triples (p_i, d_i, α_i) and test the three algorithms on the problem so generated. Each generated value is rounded off to the nearest integer, and if a negative d_i is generated, we reset $d_i = 0$, which implies adding a constant to $c_i(t)$ and therefore does not influence the final schedule.

3.3. Computational results

We generated a set of 48 problems for n = 10 and for n = 15 and 24 problems for n = 20. The four parameters, defined in section 3.2, were set at various values in order to detect their influence on the three algorithms, mentioned in section 3.1. These algorithms were coded in ALGOL 60 and run on the Control Data Cyber 73-28 computer of the SARA Computing Centre in Amsterdam. We allowed Shwimer's and our algorithm five minutes to finish each problem; lexicographic enumeration was stopped after one minute.

The computational results can be found in Tables II and III. As to the measures of performance, we remark that the *solution time* in Table I is measured in CPU-seconds, the *number of nodes* in Table III includes eliminated potential descendant nodes, a *median* was calculated only if more than half of the problems finished in time, and a *maximum* only if all problems finished in time.

The results are classified according to the value of the *average tardiness factor* t, this factor having the major influence on the performance of the algorithms. There is a significant difference between "easy" problems with t = 0.2 or t = 0.4 and "difficult" problems with t = 0.6 or t = 0.8.

On the easy problems, lexicographic enumeration is rather successful and runs quickly through large numbers of nodes. Also Shwimer's algorithm performs well, notably for n = 15 and t = 0.4. In fact, Shwimer tested his method only on problems where t = (n-1)/2n, i.e. t = 0.47 for n = 15. The new algorithm exhibits a satisfactory and steady behaviour. Both the median and maximum numbers of nodes examined by this method are significantly smaller than the numbers for the other two methods, so our lower bound is indeed more effective in pruning the search tree. For these

problems, however, it seems hardly worth-wile to spend much time on the computation of sophisticated lower bounds.

Turning to the difficult problems, we see that our algorithm is by far superior to the other algorithms. This is most clearly shown by the results for the problems with 15 or 20 jobs. Of the latter set of twelve problems, lexicographic enumeration and Shwimer's method do not finish any problem at all; our algorithm succeeds in finishing seven of them. The measures of performance become completely worthless in this situation. It is of interest to note, however, that the best solutions to unfinished problems, found by our algorithm, are better than Shwimer's. Our results seem to contradict Srinivasan's remark [21] that problems with $t = 0.65$ are the most difficult ones; problems with $t = 0.8$ are clearly the most difficult here.

We will now discuss the influence of the remaining three parameters ρ, s and t on the performance of our algorithm.

As to the *correlation* ρ, no influence at all could be demonstrated.

The *relative variation of processing times* s has a significant influence for problems with 15 or 20 jobs, as demonstrated by the sign test ($\alpha < 0.02$). For $n = 20$, eleven out of twelve problems with $s = 0.05$ were finished with a median solution time of 8 seconds, while only eight out of twelve problems with $s = 0.25$ were finished with a median of 150 seconds. On the average, 70 percent of the nodes were eliminated by the underestimate LB_r or by Elmaghraby's Lemma when $s = 0.05$, and only 40 percent when $s = 0.25$. Furthermore, the first schedule found by our algorithm, corresponding to the assignment in the root node, was at worst 1 percent from the optimum when $s = 0.05$ and at worst 20 percent when $s = 0.25$. As expected our lower bound depends heavily on s.

The *relative range of due-dates* r has a considerable influence. Problems with $r = 0.95$ are significantly easier than problems with $r = 0.20$.

Finally, we remark that both the computer programs and the test data are obtainable form the authors.

TABLE II

SOLUTION TIME

n	t	number of problems	MEDIAN			MAXIMUM		
			New Alg.	Shwimer	Lex.Enum.	New Alg.	Shwimer	Lex.Enum.
10	0.2	24	0.08	0.02	0.02	0.26	0.11	0.04
	0.6	24	0.57	0.77	1.35	3.32	42.41	47.92
15	0.2	12	0.02	0.04	0.02	0.55	0.31	0.28
	0.4	12	0.77	0.57	0.17	8.16	3.86	14.75
	0.6	12	6.29	76.68	>60	121.82	>300(3×)	>60(10×)
	0.8	12	45.61	>300	>60	85.56	>300(12×)	>60(12×)
20	0.2	6	0.78	0.17	0.09	1.19	0.32	0.24
	0.4	6	1.10	2.23	1.74	20.31	10.19	21.64
	0.6	6	180.75	>300	>60	>300(2×)	>300(6×)	>60(6×)
	0.8	6	>300	>300	>60	>300(3×)	>300(6×)	>60(6×)

TABLE III

NUMBER OF NODES

n	t	number of problems	MEDIAN			MAXIMUM		
			New Alg.	Shwimer	Lex.Enum.	New Alg.	Shwimer	Lex.Enum.
10	0.2	24	1	2	6	8	14	64
	0.6	24	56	132	3239	456	12284	96328
15	0.2	12	1	1	1	28	69	572
	0.4	12	44	86	305	541	586	36231
	0.6	12	647	13066	-	9564	-	-
	0.8	12	4532	-	-	9952	-	-
20	0.2	6	9	12	105	29	29	580
	0.4	6	25	281	3564	1206	1130	57671
	0.6	6	11105	-	-	-	-	-
	0.8	6	-	-	-	-	-	-

4. CONCLUDING REMARKS

In view of our computational results, our main conclusion clearly has to be that the problem of minimizing total weighted tardiness remains a very difficult one. *A fortiori* the same remark applies to the problem of minimizing total costs for a general cost function. Due to a lack of structure that is typical for complex combinatorial programming problems, some form of implicit enumeration still seems an obvious solution method. Our results indicate that even stronger elimination criteria and sharper lower bounds are needed to cut down the size of the search tree.

The usefulness of elimination criteria is strongly underlined by our experiments. An easy extension of our algorithm would be to check all of them in every node again. Also it may be worth-wile to look for more elimination criteria. We feel that we have thoroughly examined the possible effects of interchanging two jobs, but one may look into the effects of moving three or more jobs at a time.

The idea of computing lower bounds by solving linear assignment problems whose coefficients c_{ij} underestimate the costs of putting job J_i in position j, can be applied to a broader set of problems, e.g. to the problem of minimizing total costs in an m-machine flow shop where passing is not permitted (*see* [17]). In view of the lack of any algorithm in this area, this seems an interesting object for future research.

For the one-machine problem this lower bound has turned out to be very useful. It could be strengthened by considering only those solutions to the assignment problem that respect known precedence relations. It is difficult to predict the effectiveness of this approach, since the resulting integer programming problem seems rather complicated, and the precedence relations are observed already in the computation of the present cost coefficients.

None the less, it seems necessary to develop a fundamentally stronger lower bound. Especially, very sharp bounds should be used in the upper levels of the search tree, where pruning may lead to large reductions in the number of potentially optimal solutions. As we move down the tree, pruning leads to smaller reductions, and simpler lower bounds combined with more extensive enumeration become more attractive. This suggests the use of lower bounds of varying computational complexity throughout the tree: a

gliding lower bound. We tried to apply this idea in our algorithm by using lexicographic enumeration in the seven deepest levels of the tree. This led to a disappointingly small decrease in computation time, but the idea could become useful in the future.

In spite of all the work done so far, the problem of minimizing total costs in one-machine scheduling is likely to remain a challenge to researchers for a long time to come.

ACKNOWLEDGEMENT

We would like to acknowledge B. Dorhout's cooperation in making his assignment code available to us.

REFERENCES

1. K.R. BAKER AND J.B. MARTIN, "An Experimental Comparison of Solution Algorithms for the Single-Machine Tardiness Problem," Technical Report 72-6, Department of Industrial Engineering, University of Michigan, Ann Arbour, Mich., 1972.
2. E.M.L. BEALE, "An Algorithm for Solving the Transportation Problem when the Shipping Cost over Each Route is Convex," *Naval Res. Log. Quart.* 6, 43-56 (1959).
3. A. CHARNES AND W.W. COOPER, "The Stepping-Stone Method of Explaining Linear Programming Calculations in Transportation Problems," *Management Sci.* 1, 49-69 (1954).
4. R.W. CONWAY, W.L. MAXWELL, AND L.W. MILLER, *Theory of Scheduling*, Addison-Wesley, Reading, Mass., 1967.
5. B. DORHOUT, "Het Lineaire Toewijzingsprobleem: Vergelijking van Algoritmen," Report BN 21, Mathematisch Centrum, Amsterdam, 1973.
6. E.W. DIJKSTRA, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik* 1, 269-271 (1959).
7. S.E. ELMAGHRABY, "The One-Machine Sequencing Problem with Delay Costs," *J. Indust. Eng.* 19, 105-108 (1968).
8. H. EMMONS, "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," *Opns. Res.* 17, 701-715 (1969).
9. L.R. FORD, JR., AND D.R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, N.J., 1962.
10. L. GELDERS AND P.R. KLEINDORFER, "Coordinating Aggregate and Detailed Scheduling Decisions in the One-Machine Job Shop: Part I. Theory," *Opns. Res.* 22, 46-60 (1974).
11. L. GELDERS AND P.R. KLEINDORFER, "Coordinating Aggregate and Detailed Scheduling in the One-Machine Job Shop: II - Computation and Structure," Report I/72-33, International Institute of Management, Berlin, 1972.
12. M. HELD AND R.M. KARP, "A Dynamic Programming Approach to Sequencing Problems," *J. Soc. Indust. and Appl. Math.* 10, 196-210 (1962).
13. R.M. KARP, "Reducibility among Combinatorial Problems," pp. 85-103 in R.E. MILLER AND J.W. THATCHER (eds.), *Complexity of Computer Computations*, Plenum Press, New York, N.Y., 1972.

14. E.L. LAWLER, "On Scheduling Problems with Deferral Costs," *Management Sci.* 11, 280-288 (1964).

15. E.L. LAWLER, "Optimal Sequencing of a Single Machine Subject to Precedence Constraints," *Management Sci.* 19, 544-546 (1973).

16. J.K. LENSTRA, "Recursive Algorithms for Enumerating Subsets, Lattice-Points, Combinations and Permutations," Report BW 28, Mathematisch Centrum, Amsterdam, 1973.

17. A.H.G. RINNOOY KAN, "The Machine Scheduling Problem," Report BW 27, Mathematisch Centrum, Amsterdam, 1973; also Report R/73/4, Interfaculty for Graduate Studies in Management, Delft, 1973.

18. A. SCHILD AND I.J. FREDMAN, "Scheduling Tasks with Deadlines and Non-linear Loss Functions," *Management Sci.* 9, 73-81 (1962).

19. J. SHWIMER, "On the *N*-Job, One-Machine, Sequence-Independent Scheduling Problem with Tardiness Penalties: a Branch-and-Bound Solution," *Management Sci.* 18, B301-313 (1972).

20. W.E. SMITH, "Various Optimizers for Single-Stage Production," *Naval Res. Log. Quart.* 3, 59-66 (1956).

21. V. SRINIVASAN, "A Hybrid Algorithm for the One-Machine Sequencing Problem to Minimize Total Tardiness," *Naval Res. Log. Quart.* 18, 317-327 (1971).

22. Y. TABOURIER, "Un Algorithme pour le Problème d'Affectation," *Revue française d'Automatique, Informatique et Recherche Operationelle* 6, 3-15 (1972).

23. N. TOMIZAWA, "On Some Techniques Useful for Solution of Transportation Network Problems," *Networks* 1, 173-194 (1971).

ALGOL 60 PROCEDURES

A. New algorithm

```
"INTEGER" "PROCEDURE" MSP SUMAKTK BB
(N, P, D, A, ZSTAR, XSTAR, NN, NF, NP, NC, NE, EL);
"VALUE" N,ZSTAR; "INTEGER" N,ZSTAR,NN,NF,NP,NC,NE,EL;
"INTEGER" "ARRAY" P,D,A,XSTAR;
"COMMENT" SCHEDULING N JOBS ON 1 MACHINE
          * NON-DECREASING PROCESSING TIMES : P[1:N]
          * DUE DATES : D[1:N]
          * WEIGHTS : A[1:N]
          MINIMIZING THE SUM OF THE WEIGHTED TARDINESSES
          * UPPERBOUND : ZSTAR
          * MINIMUM : MSP SUMAKTK BB
          * IF UPPERBOUND > MINIMUM THEN OPTIMAL SEQUENCE : XSTAR[1:N]
          BY BRANCH-AND-BOUND
          * NN IS THE NUMBER OF NODES IN THE SEARCH TREE
          * NF NODES LEAD TO BRANCHING
          * NP NODES ARE ELIMINATED BY SOLVING AN ASSIGNMENT PROBLEM
          * NC NODES ARE ELIMINATED BY A DUAL UNDERESTIMATE
          * NE NODES ARE ELIMINATED BY ELMAGHRABY'S LEMMA
          * EL TIMES ELMAGHRABY'S LEMMA IS APPLIED;
"BEGIN" "INTEGER" I,J,H,K,I0,J0,T,CIJ,UI,MIN,UP,PI,DI,AI,NAI,NBI,
  PAI,PBI; "BOOLEAN" CYCLE;
  "INTEGER" "ARRAY" ELM,X,Y,U,V,NA,NB,PA,PB,DIS,NEXT[1:N],
  LAB,SUC,PRE[0:N],C,PREC[1:N,1:N];

  "PROCEDURE" NEWSOL(Z); "VALUE" Z; "INTEGER" Z;
  "BEGIN" "INTEGER" J; ZSTAR:= Z;
    "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO" XSTAR[J]:= X[J]
  "END" NEWSOL;

  "PROCEDURE" VEC IND QSORT(A, IND, LO, UP); "CODE" 11021;
  "COMMENT" VEC IND QSORT REARRANGES THE CONTENTS OF IND[LO],---,IND[UP]
  SUCH THAT A[IND[LO]],---,A[IND[UP]] HAVE ASCENDING ORDER,
  USING ALGORITHM 402, COMM.ACM, 13,693(1970);

  "PROCEDURE" VEC2    QSORT(A, IND, LO, UP); "CODE" 11024;
  "COMMENT" VEC2 QSORT REARRANGES THE PAIRS OF NUMBERS (A[LO],IND[LO]),
  ---,(A[UP],IND[UP]) INTO ASCENDING ORDER OF THE FIRST MEMBERS,
  USING THE SAME SORTING ALGORITHM AS VEC IND QSORT;
```

```
"PROCEDURE" PRECEDENCE RELATIONS(N); "VALUE" N; "INTEGER" N;
"BEGIN" "INTEGER" DPMAX,T; "BOOLEAN" "ARRAY" AFT,BEF[1:N];

  "PROCEDURE" TRANS(J,K); "VALUE" J,K; "INTEGER" J,K;
  "BEGIN" "INTEGER" I,L;
    "FOR" I:= PRE[0],PRE[I] "WHILE" I>0 "DO"
    "IF" I=K ! PREC[K,I]=-1 "THEN"
    "FOR" L:= J,PREC[J,L] "WHILE" L>0 "DO"
    "IF" PREC[I,L]=0 "THEN"
    "BEGIN" PREC[L,I]:= -1; PREC[I,L]:= PREC[I,I]; PREC[I,I]:= L;
      PA[L]:= T:= PA[L]-P[I]; AFT[L]:= DPMAX >= T;
      PB[I]:= T:= PB[I]+P[L];
      "IF" T > D[I] "THEN" CYCLE:= BEF[I]:= "TRUE"
    "END"
  "END" TRANS;

  DPMAX:= 0; I:= SUC[0]; T:= PA[I]; CYCLE:= "FALSE";
  "FOR" I:= I, SUC[I] "WHILE" I>0 "DO"
  "BEGIN" BEF[I]:= "TRUE"; AFT[I]:= CYCLE; PBI:= D[I]+P[I];
    "IF" PBI > DPMAX "THEN"
    "BEGIN" DPMAX:= PBI; CYCLE:= DPMAX >= T "END"
  "END";
  "FOR" I:= 0, 0 "WHILE" CYCLE "DO"
  "BEGIN" CYCLE:= "FALSE";
    "FOR" I:= SUC[I] "WHILE" I>0 "DO" "IF" AFT[I] ! BEF[I] "THEN"
    "BEGIN" AI:= A[I]; DI:= D[I]; PAI:= PA[I];
      PBI:= PB[I]; "IF" PBI < DI "THEN" PBI:= DI;
      "FOR" J:= LAB[I] "WHILE" BEF[I] "DO"
      "BEGIN" BEF[I]:= "FALSE";
        "FOR" J:= PRE[J] "WHILE" J>0 "DO"
        "IF" PREC[I,J]^=0 "THEN" "ELSE"
        "IF" A[J]>=AI & D[J]<=PBI "THEN"
        "BEGIN" TRANS(J,I);
          "IF" BEF[I] "THEN" PBI:= PB[I] "ELSE" CYCLE:= "TRUE"
        "END"
      "END";
      "FOR" J:= 0 "WHILE" AFT[I] "DO"
      "BEGIN" AFT[I]:= "FALSE";
        "FOR" J:= SUC[J] "WHILE" J>0 "DO"
        "IF" PREC[I,J]^=0 "THEN" "ELSE"
        "IF" "IF" D[J]>=PAI "THEN" "TRUE" "ELSE" "IF" J>I "THEN"
        "FALSE" "ELSE" A[J]<=AI & D[J]>=DI & D[J]+P[J]>=PAI "THEN"
        "BEGIN" TRANS(I,J); PAI:= PA[I] "END"
      "END"
    "END"
  "END"
"END" PRECEDENCE RELATIONS;
```

```
"PROCEDURE" NODE(M,Q,LB,CS,LF,U,V);
"VALUE" M,Q,LB,CS,LF,U,V; "INTEGER" M,Q,LB,CS,LF;
"INTEGER" "ARRAY" U,V;
"BEGIN" "INTEGER" L,LEV,I0,K,H; L:= M;
  "FOR" H:= LF "STEP" -1 "UNTIL" 1 "DO"
  "BEGIN" I:= ELM[H];
    "IF" Y[I] > L "THEN" LF:= LF-1 "ELSE"
    "IF" D[I] < Q "THEN" H := 0    "ELSE"
    "BEGIN" EL:= EL+1; LB:= LB-U[I]*V[L];
      "IF" LB >= ZSTAR "THEN" "BEGIN" NE:= NE+1; "GOTO" OUT "END";
      J:= Y[I]; "IF" J < L "THEN"
      "BEGIN" K:= X[L]; "IF" K > 0 "THEN" Y[K]:= 0;
        X[L]:= I; "IF" J > 0 "THEN" X[J]:= 0; Y[I]:= L
      "END";
      J:= PRE[I]; SUC[J]:= K:= SUC[I]; PRE[K]:= J;
      J:= I; "FOR" J:= PREC[I,J] "WHILE" J>0 "DO" NA[J]:= NA[J]-1;
      Q:= Q-P[I]; L:= L-1; LF:= H-1
    "END"
  "END";
  "IF" L = 0 "THEN"
  "BEGIN" NEWSOL(CS); NE:= NE+1 "END" "ELSE"
  "BEGIN" "INTEGER" "ARRAY" LAST,LBR[-L:-1];
    NP:= NP+1; LB:= ZSTAR-LB; I0:= 1; PI:= 0;
    "FOR" I:= SUC[0],SUC[I] "WHILE" I>0 "DO"
    "BEGIN" PBI:= PB[I]; AI:= A[I]; NBI:= J0:= NB[I];
      MIN:= -V[NBI]; J:= 0;
      "FOR" J:= SUC[J] "WHILE" J>0 "DO" "IF" PREC[I,J]=0 "THEN"
      "BEGIN" NBI:= NBI+1; PBI:= PBI+P[J];
        C[I,NBI]:= CIJ:= "IF" PBI > 0 "THEN" AI*PBI "ELSE" 0;
        CIJ:= CIJ-V[NBI]; "IF" CIJ < MIN "THEN"
        "BEGIN" MIN:= CIJ; J0:= NBI "END"
      "END";
      "FOR" NBI:= NBI+1 "STEP" 1 "UNTIL" L "DO"
      "IF" C[I,NBI]="10 "THEN" NBI:= L "ELSE" C[I,NBI]:= "10;
      H:= MIN-U[I]; "IF" H > 0 "THEN"
      "BEGIN" LB:= LB-H; "IF" LB <= 0 "THEN" "GOTO" OUT;
        U[I]:= MIN
      "END";
      J:= Y[I]; "IF" "IF" J=0 "THEN" "FALSE" "ELSE" MIN+V[J]<C[I,J]
      "THEN" Y[I]:= X[J]:= J:= 0;
      "IF" J=0 & X[J0]=0 "THEN" "BEGIN" X[J0]:= I; Y[I]:= J0 "END"
    "END";
    "FOR" J:= L "STEP" -1 "UNTIL" 1 "DO" "IF" X[J]=0 "THEN"
    "BEGIN" MIN:= "9;
      "FOR" I:= SUC[0],SUC[I] "WHILE" I>0 "DO"
      "BEGIN" CIJ:= C[I,J]-U[I]; "IF" CIJ <= MIN "THEN"
        "BEGIN" I0:= I;
          MIN:= "IF" Y[I0]=0 "THEN" CIJ-1 "ELSE" CIJ
        "END"
      "END";
      T:= "IF" Y[I0]=0 "THEN" MIN+1 "ELSE" MIN; H:= T-V[J];
      "IF" H > 0 "THEN"
      "BEGIN" LB:= LB-H; "IF" LB <= 0 "THEN" "GOTO" OUT;
        V[J]:= T
      "END";
      "IF" T > MIN "THEN" "BEGIN" X[J]:= I0; Y[I0]:= J "END"
    "END";
    "FOR" I0:= PRE[0],PRE[I0] "WHILE" I0>0 "DO" "IF" Y[I0]=0 "THEN"
    "BEGIN" MIN:= "9; NBI:= NB[I0]; NAI:= L-NA[I0];
      "FOR" J:= NBI "STEP" 1 "UNTIL" NAI "DO"
      "BEGIN" DIS[J]:= DI:= C[I0,J]-V[J]; NEXT[J]:= J; LAB[J]:= I0;
        "IF" DI <= MIN "THEN"
        "BEGIN" J0:= J;
          MIN:= "IF" X[J]=0 "THEN" DI-1 "ELSE" DI
        "END"
      "END";
      UI:= "IF" X[J0]=0 "THEN" MIN+1 "ELSE" MIN; H:= UI-U[I0]+PI;
```

```
      U[I0]:= UI; "IF" H > 0 "THEN"
      "BEGIN" LB:= LB-H; "IF" LB <= 0 "THEN" "GOTO" OUT "END";
      "IF" UI > MIN "THEN"
      "BEGIN" Y[I0]:= J0; X[J0]:= I0; "GOTO" NEXTI "END";
      "FOR" J:= NBI-1 "STEP" -1 "UNTIL" 1,
      NAI+1 "STEP" 1 "UNTIL" L "DO"
      "BEGIN" DIS[J]:= "9; NEXT[J]:= J "END";
      UP:= L;
      "FOR" I:= X[J0],X[NEXT[J0]] "WHILE" I>0 & MIN^="8 "DO"
      "BEGIN" NEXT[J0]:= NEXT[UP]; UP:= UP-1; H:= U[I]-MIN; MIN:= "8;
        "FOR" K:= 1 "STEP" 1 "UNTIL" UP "DO"
        "BEGIN" J:= NEXT[K]; CIJ:= C[I,J]-H-V[J]; DI:= DIS[J];
          "IF" CIJ < DI "THEN"
          "BEGIN" DIS[J]:= DI:= CIJ; LAB[J]:= I "END";
          "IF" DI <= MIN "THEN"
          "BEGIN" J0:= K;
            MIN:= "IF" X[J]=0 "THEN" DI-1 "ELSE" DI
          "END"
        "END"
      "END";
      MIN:= MIN+1-UI; "IF" MIN > 0 "THEN"
      "BEGIN" LB:= LB-MIN; "IF" LB <= 0 "THEN" "GOTO" OUT;
        PI:= PI+MIN;
        "FOR" J:= 1 "STEP" 1 "UNTIL" L "DO"
        "BEGIN" H:= DIS[J]-UI; "IF" MIN < H "THEN" H:= MIN;
          V[J]:= V[J]+H; I:= X[J]; "IF" I > 0 "THEN" U[I]:= U[I]-H
        "END"
      "END";
      "FOR" J:= NEXT[J0],J0 "WHILE" J>0 "DO"
      "BEGIN" X[J]:= I:= LAB[J]; J0:= Y[I]; Y[I]:= J "END";
    NEXTI:
    "END";
    T:= Q; I:= X[L]; CIJ:= ZSTAR-CS-C[I,L];
    "FOR" J:= L-1 "STEP" -1 "UNTIL" 2 "DO"
    "BEGIN" T:= T-P[I]; I:= X[J]; H:= T-D[I]; "IF" H > 0 "THEN"
      "BEGIN" CIJ:= CIJ-A[I]*H;
        "IF" CIJ <= 0 "THEN" "GOTO" BRANCH
      "END"
    "END";
    NEWSOL(ZSTAR-CIJ);
    LB:= LB-CIJ; "IF" LB <= 0 "THEN" "GOTO" OUT;

  BRANCH; NF:= NF+1; LB:= LB+V[L]; LEV:= 0;
    "FOR" I0:= PRE[0],PRE[I0] "WHILE" I0>0 "DO" "IF" NA[I0]=0 "THEN"
    "BEGIN" NN:= NN+1; CIJ:= LB-C[I0,L]+U[I0]; "IF" CIJ > 0 "THEN"
      "BEGIN" LEV:= LEV-1; LAST[LEV]:= I0; LBR[LEV]:= ZSTAR-CIJ "END"
    "END";
    "IF" LEV=0 "THEN" "GOTO" OUT;
    VEC2 QSORT(LBR,LAST,LEV,-1); LB:= L-1;
    "FOR" LEV:= LEV "STEP" 1 "UNTIL" -1 "DO"
    "IF" LBR[LEV] >= ZSTAR "THEN" LEV:= 0 "ELSE"
    "BEGIN" I:= I0:= LAST[LEV]; J:= Y[I0]; "IF" J < L "THEN"
      "BEGIN" Y[X[L]]:= 0; X[L]:= I0; Y[I0]:= L;
        "IF" J > 0 "THEN" X[J]:= 0
      "END";
      "FOR" I:= PREC[I0,I] "WHILE" I>0 "DO" NA[I]:= NA[I]-1;
      H:= PRE[I0]; SUC[H]:= K:= SUC[I0]; PRE[K]:= H;
      NODE(LB,Q-P[I0],LBR[LEV],CS+C[I0,L],LF,U,V);
      I:= I0; "FOR" I:= PREC[I0,I] "WHILE" I>0 "DO" NA[I]:= NA[I]+1;
      SUC[H]:= PRE[K]:= I0
    "END"
  "END";
OUT; "FOR" L:= L+1 "STEP" 1 "UNTIL" M "DO"
  "BEGIN" I:= X[L]; SUC[PRE[I]]:= PRE[SUC[I]]:= J:= I;
    "FOR" J:= PREC[I,J] "WHILE" J>0 "DO" NA[J]:= NA[J]+1
  "END"
"END" NODE;
```

```
  NN:= 1; NF:= NP:= NE:= UP:= T:= 0; H:= N;
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" ELM[I]:= I; NA[I]:= 0; T:= T+P[I] "END";
  VEC IND QSORT(D,ELM,1,N);
  "FOR" J:= N "STEP" -1 "UNTIL" 1 "DO"
  "BEGIN" I:= ELM[J]; "IF" D[I] < T "THEN" J:= 0 "ELSE"
    "BEGIN" T:= T-P[I]; XSTAR[N]:= NA[I]:= I; N:= N-1 "END"
  "END";
  EL:= H-N; "IF" N=0 "THEN"
  "BEGIN" NE:= 1; ZSTAR:= 0 "END" "ELSE"
  "BEGIN" DI:= "10;
    "FOR" I:= H "STEP" -1 "UNTIL" 1 "DO" "IF" NA[I]=0 "THEN"
    "BEGIN" PB[I]:= PI:= P[I]; PA[I]:= T;
      "IF" PI < DI "THEN" "BEGIN" DI:= PI; J0:= UP "END";
      SUC[I]:= UP; PRE[UP]:= UP:= I;
      LAB[I]:= "IF" PRE[J0]=I "THEN" I "ELSE" J0;
      "FOR" J:= 1 "STEP" 1 "UNTIL" H "DO" PREC[I,J]:= 0;
      PREC[I,I]:= -2
    "END";
    SUC[0]:= UP; PRE[UP]:= 0;
    PRECEDENCE RELATIONS(H);
    CIJ:= 0;
    "FOR" I:= SUC[0],SUC[I] "WHILE" I>0 "DO"
    "BEGIN" NBI:= 1; NAI:= J:= 0;
      "FOR" J:= SUC[J] "WHILE" J>0 "DO"
      "IF" PREC[I,J]=-1 "THEN"
      "BEGIN" C[I,N-NAI]:= "10; NAI:= NAI+1 "END" "ELSE"
      "IF" PREC[I,J]> 0 "THEN"
      "BEGIN" C[I,NBI]:= "10; NBI:= NBI+1 "END";
      NB[I]:= NBI; NA[I]:= U[I]:= NAI;
      PB[I]:= PBI:= PB[I]-D[I]; C[I,NBI]:= 0; "IF" PBI > 0 "THEN"
      "BEGIN" PB[I]:= 0; D[I]:= D[I]+PBI; CIJ:= CIJ+A[I]*PBI "END"
    "END";
    "FOR" I0:= N "STEP" -1 "UNTIL" 1 "DO"
    "BEGIN" DI:= V[I0]:= 0;
      "FOR" K:= I0 "STEP" -1 "UNTIL" 1 "DO"
      "BEGIN" J:= ELM[K]; "IF" U[J]=0 & D[J]>DI "THEN"
        "BEGIN" DI:= D[J]; H:= K "END"
      "END";
      I:= ELM[H]; ELM[H]:= ELM[I0]; ELM[I0]:= X[I0]:= J:= I; Y[I]:= I0;
      "FOR" J:= PREC[I,J] "WHILE" J>0 "DO" U[J]:= U[J]-1
    "END";
    NODE(N,T,CIJ,CIJ,N,U,V)
  "END";
  NC:= NN-NP-NE; NP:= NP-NF;
  MSP SUMAKTK BB:= ZSTAR
"END" MSP SUMAKTK BB;
```

B. Shwimer's algorithm

```
"INTEGER" "PROCEDURE" MSP SUMAKTK BB SHWIMER
(N, P, D, A, ZSTAR, XSTAR, NN, NA, NB, NC, EL);
"VALUE" N,ZSTAR; "INTEGER" N,ZSTAR,NN,NA,NB,NC,EL;
"INTEGER" "ARRAY" P,D,A,XSTAR;
"COMMENT" SCHEDULING N JOBS ON 1 MACHINE
          * NON-DECREASING PROCESSING TIMES : P[1:N]
          * DUE DATES : D[1:N]
          * WEIGHTS : A[1:N]
          MINIMIZING THE SUM OF THE WEIGHTED TARDINESSES
          * UPPERBOUND : ZSTAR
          * MINIMUM : MSP SUMAKTK BB
          * IF UPPERBOUND > MINIMUM THEN OPTIMAL SEQUENCE : XSTAR[1:N]
          BY A BRANCH-AND-BOUND ALGORITHM OF J.SHWIMER,
          MANAGEMENT SCI.18,B301-313(1972)
          * NN IS THE NUMBER OF NODES IN THE SEARCH TREE
          * NA NODES ARE ELIMINATED BY A LOWER BOUND CALCULATION
          * NB NODES LEAD TO BRANCHING
          * NC NODES PRODUCE A BETTER SOLUTION
          * EL TIMES ELMAGHRABY'S LEMMA IS APPLIED;
"BEGIN" "INTEGER" I,J,H,K,I1,PI,T,AI,MAXH,MAXT,LMAX,IMAX,IMIN,
  CIJ,A1,A2,Q,DI,J1;
  "INTEGER" "ARRAY" X,LATE,NOTLAST[1:N],SUC,PRE[0:N],PREC[1:N,1:N];

  "PROCEDURE" NEWSOL(Z); "VALUE" Z; "INTEGER" Z;
  "BEGIN" "INTEGER" J; ZSTAR:= Z;
    "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO" XSTAR[J]:= X[J]
  "END" NEWSOL;

  "PROCEDURE" VEC IND QSORT(A, IND, LO, UP); "CODE" 11021;
  "PROCEDURE" VEC2    QSORT(A, IND, LO, UP); "CODE" 11024;
  "COMMENT" SEE PROCEDURE MSP SUMAKTK BB;

  "INTEGER" "PROCEDURE" LOWERBOUND SHWIMER(Q);
  "VALUE" Q; "INTEGER" Q;
  "BEGIN" NN:= NN+1;
    T:= MAXH:= 0; CIJ:= A1:= "15; IMAX:= SUC[0];
    "FOR" I:= IMAX,SUC[I] "WHILE" I>0 "DO"
    "BEGIN" T:= T+P[I]; LATE[I]:= T-D[I];
      AI:= A[I]; "IF" AI < A1 "THEN"
      "BEGIN" A2:= A1; A1:= AI; IMIN:= I "END" "ELSE"
      "IF" AI < A2 "THEN" A2:= AI
    "END";
    "FOR" I:= IMAX,SUC[I] "WHILE" I>0 "DO"
    "BEGIN" "IF" I=IMAX "THEN"
      "BEGIN" J:= I; LMAX:= 0;
        "FOR" J:= SUC[J] "WHILE" J>0 "DO"
        "IF" LATE[J] > LMAX "THEN"
        "BEGIN" LMAX:= LATE[J]; IMAX:= J "END"
      "END";
      MAXT:= LMAX-P[I]; T:= Q-D[I];
      MAXT:=  ("IF" T<=0 "THEN" 0 "ELSE" A[I]*T) +
              ("IF" MAXT>MAXH "THEN" MAXT "ELSE" MAXH) *
              ("IF" I=IMIN "THEN" A2 "ELSE" A1);
      "IF" MAXT < CIJ "THEN" CIJ:= MAXT;
      "IF" LATE[I] > MAXH "THEN" MAXH:= LATE[I]
    "END";
    LOWERBOUND SHWIMER := CIJ
  "END" LOWERBOUND SHWIMER;
```

```
  "PROCEDURE" NODE(M,Q,Z);
  "VALUE" M,Q,Z; "INTEGER" M,Q,Z;
  "BEGIN" "INTEGER" L,L1,LEV,I0,K,H;
    NB:= NB+1; L:= M; I:= PRE[0];
    "FOR" I:= I "WHILE" "IF" I=0 "THEN" "FALSE" "ELSE" D[I] >= Q "DO"
    "BEGIN" X[L]:= I; EL:= EL+1; L:= L-1; Q:= Q-P[I];
      PRE[0]:= I:= PRE[I]; SUC[I]:= 0
    "END";
    "IF" L <= 1 "THEN"
    "BEGIN" "IF" L=1 "THEN"
      "BEGIN" X[1]:= I:= SUC[0]; Z:= Z+A[I]*(Q-D[I]) "END";
      "IF" Z < ZSTAR "THEN"
      "BEGIN" NEWSOL(Z); NC:= NC+1 "END"
    "END" "ELSE"
    "BEGIN" "INTEGER" "ARRAY" LAST,LB[-L:-1];
      LEV:= 0;
      "FOR" I0:= PRE[0],PRE[I0] "WHILE" I0 > 0 "DO"
      "IF" NOTLAST[I0] < NB "THEN"
      "BEGIN" I:= I0; "FOR" I:= PREC[I0,I] "WHILE" I>0 "DO"
        NOTLAST[I]:= NB
      "END";
      "FOR" I0:= PRE[0],H "WHILE" I0 > 0 "DO"
      "BEGIN" H:= PRE[I0];"IF" NOTLAST[I0] < NB "THEN"
        "BEGIN" SUC[H]:= K:= SUC[I0]; PRE[K]:= H;
        T:= LOWERBOUND SHWIMER(Q-P[I0])+Z+A[I0]*(Q-D[I0]);
        "IF" T < ZSTAR "THEN"
        "BEGIN" LEV:= LEV-1; LAST[LEV]:= I0; LB[LEV]:= T "END";
        SUC[H]:= PRE[K]:= I0
        "END"
      "END";
      "IF" LEV < 0 "THEN"
      "BEGIN" L1:= L-1;
        VEC2 QSORT(LB,LAST,LEV,-1);
        "FOR" LEV:= LEV "STEP" 1 "UNTIL" -1 "DO"
        "IF" LB[LEV] >= ZSTAR "THEN" LEV:= 0 "ELSE"
        "BEGIN" I0:= X[L]:= LAST[LEV];
          H:= PRE[I0]; SUC[H]:= K:= SUC[I0]; PRE[K]:= H;
          NODE(L1,Q-P[I0],Z+A[I0]*(Q-D[I0]));
          SUC[H]:= PRE[K]:= I0
        "END"
      "END"
    "END";
    "FOR" L:= L+1 "STEP" 1 "UNTIL" M "DO"
    "BEGIN" I:= X[L]; SUC[PRE[I]]:= PRE[SUC[I]]:= I "END"
  "END" NODE;

  NN:= 1; NB:= NC:= EL:= Q:= J:= CIJ:= 0;
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" X[I]:= I; SUC[I]:= "8*D[I]-A[I] "END";
  VEC IND QSORT(SUC,X,1,N);
  "FOR" K:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" I:= X[K]; PRE[I]:= J; SUC[J]:= J:= I; Q:= Q+P[I];
    H:= Q-D[I]; "IF" H > 0 "THEN" CIJ:= CIJ+A[I]*H;
    AI:= A[K]; PI:= P[K]; DI:= D[K]; I1:= K;
    "FOR" H:= K-1 "STEP" -1 "UNTIL" 1 "DO"
    "IF" A[H]>=AI "THEN" LATE[I1]:= I1:= H;
    LATE[I1]:= 0; I1:= J1:= K;
    "FOR" J1:= LATE[J1]"WHILE" J1>0 "DO"
    "IF" D[J1]<=DI "THEN" PREC[K,I1]:= I1:= J1;
    PREC[K,I1]:= NOTLAST[I]:= 0
  "END";
  SUC[I]:= 0; PRE[0]:= I;
  "IF" CIJ < ZSTAR "THEN" NEWSOL(CIJ);
  NODE(N,Q,0);
  NA:= NN-NB; NB:= NB-NC;
  MSP SUMAKTK BB SHWIMER := ZSTAR
"END" MSP SUMAKTK BB SHWIMER;
```

C. Lexicographic enumeration algorithm

```
"INTEGER" "PROCEDURE" MSP SUMAKTK LEX
(N,P,D,A,ZSTAR,XSTAR,NA,NB,NC,EL); "VALUE" N,ZSTAR;
"INTEGER" N,ZSTAR,NA,NB,NC,EL; "INTEGER" "ARRAY" P,D,A,XSTAR;
"COMMENT" SCHEDULING N JOBS ON 1 MACHINE
          * PROCESSING TIMES: P[1:N]
          * DUE-DATES: D[1:N]
          * WEIGHTS: A[1:N]
          MINIMIZING THE SUM OF WEIGHTED TARDINESSES
          * UPPERBOUND: ZSTAR
          * MINIMUM: MSP SUMAKTK LEX
          * IF UPPERBOUND > MINIMUM THEN OPTIMAL SEQUENCE: XSTAR[1:N]
          BY BOUNDED LEXICOGRAPHIC ENUMERATION
          * NA NODES ARE ELIMINATED
          * NB NODES LEAD TO BRANCHING
          * NC NODES PRODUCE A BETTER SOLUTION
          * EL TIMES ELMAGHRABY'S LEMMA IS APPLIED;
"BEGIN" "INTEGER" Q, H; "INTEGER" "ARRAY" X[1:N];

  "PROCEDURE" VEC IND QSORT(A, IND, LO, UP); "CODE" 11021;
  "COMMENT" SEE PROCEDURE MSP SUMAKTK BB;

  "PROCEDURE" NODE(M,Q,Z); "VALUE" M,Q,Z; "INTEGER" M,Q,Z;
  "BEGIN" "INTEGER" K, L, XM;
    "FOR" H:= M "STEP" -1 "UNTIL" 1 "DO"
    "BEGIN" XM:= X[M];
      "IF" D[XM] < Q "THEN" H:= 0 "ELSE"
      "BEGIN" EL:= EL + 1;
        M:= M - 1; Q:= Q - P[XM]
      "END"
    "END";
    "IF" M = 0 "THEN"
    "BEGIN" NC:= NC + 1;
      ZSTAR:= Z;
      "FOR" H:= N "STEP" -1 "UNTIL" 1 "DO" XSTAR[H]:= X[H]
    "END"       "ELSE"
    "BEGIN" NB:= NB + 1;
      L:= M - 1;
      "FOR" K:= M "STEP" -1 "UNTIL" 1 "DO"
      "BEGIN" X[M]:= H:= X[K]; X[K]:= XM; XM:= H;
        H:= Z + A[XM] * (Q - D[XM]);
        "IF" H < ZSTAR "THEN" NODE(L,Q - P[XM],H) "ELSE" NA:= NA + 1
      "END";
      "FOR" K:= L "STEP" -1 "UNTIL" 1 "DO" X[K + 1]:= X[K]; X[1]:= XM
    "END"
  "END";

  NA:= NB:= NC:= EL:= Q:= 0;
  "FOR" H:= N "STEP" -1 "UNTIL" 1 "DO"
  "BEGIN" X[H]:= H;
    Q:= Q + P[H]
  "END";
  VEC IND QSORT(D,X,1,N);
  "IF" ZSTAR > 0 "THEN" NODE(N,Q,0);
  MSP SUMAKTK LEX:= ZSTAR
"END" MSP SUMAKTK LEX;
```