Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs

C. N. Potts, L. N. Van Wassenhove,

To cite this article:
C. N. Potts, L. N. Van Wassenhove, (1988) Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs. Management Science 34(7):843-858. https://doi.org/10.1287/mnsc.34.7.843

MANAGEMENT SCIENCE
Vol. 34, No. 7, July 1988
Printed in U.S.A

ALGORITHMS FOR SCHEDULING A SINGLE MACHINE TO MINIMIZE THE WEIGHTED NUMBER OF LATE JOBS*

C. N. POTTS AND L. N. VAN WASSENHOVE
Faculty of Mathematical Studies, University of Southampton, Southampton, England
Afdeling Industrieel Beleid, Katholieke Universiteit Leuven, Belgium

This paper considers the problem of scheduling n jobs, each having a processing time, a due date and a weight, on a single machine to minimize the weighted number of late jobs. An $O(n \log n)$ algorithm is given for solving the linear programming problem obtained by relaxing the integrality constraints in a zero-one programming formulation of the problem. This linear programming lower bound is used in a reduction algorithm that eliminates jobs from the problem. Also, a branch and bound algorithm that uses the linear programming lower bound is proposed. Computational results with branch and bound algorithms that use this and other lower bounds and with a dynamic programming algorithm for problems with up to 1000 jobs are given.

(SCHEDULING; REDUCTION; DYNAMIC PROGRAMMING; BRANCH AND BOUND)

1. Introduction

The single machine weighted number of late jobs problem may be stated as follows. Each of n jobs (numbered $1, \ldots, n$) is to be processed without interruption on a single machine which can handle only one job at a time. Job j ($j = 1, \ldots, n$) becomes available for processing at time zero, requires an integer *processing time* p_j and has an integer *due date* d_j, where $p_j \leqslant d_j$. Also, associated with each job j is a nonnegative *weight* w_j which is the penalty incurred if it is completed after its due date d_j. The objective is to sequence the jobs so that the total penalty, i.e., the weighted number of late jobs, is minimized.

It is assumed henceforth that the jobs have been renumbered according to their earliest due date (EDD) ordering so that $d_1 \leqslant \cdots \leqslant d_n$. When the jobs have been scheduled, the completion time C_j of each job j can be determined. If $C_j \leqslant d_j$ then job j is *early*; otherwise $C_j > d_j$ and job j is *late*. The solution of the problem essentially involves specifying the sets of early and late jobs: the early jobs are sequenced first in EDD order (Lawler and Moore 1969) followed by the late jobs sequenced arbitrarily. If all jobs of a set E are early when sequenced in EDD order, then E is a *feasible set of early jobs*.

Lawler and Moore derive a pseudopolynomial dynamic programming algorithm which requires $O(n \min \{\sum_j p_j, \max_j \{d_j\}\})$ time. When all weights are integer, this formulation is generalized by Sahni (1976) to produce an algorithm which requires $O(n \min \{\sum_j p_j, \sum_j w_j, \max_j \{d_j\}\})$ time. Since the performance in terms of computer time and storage requirements of both algorithms is highly dependent on the size of the input data, they do not necessarily provide practical solution methods. No computational experience with either algorithm is reported in the literature.

Villarreal and Bulfin (1983) propose branch and bound algorithms based upon two lower bounding procedures and a dominance theorem. Computational experience indicates that their algorithms can solve 50-job problems using a few seconds of computer time on a CDC CYBER 175.

Some special cases of the weighted number of late jobs problem deserve mention. For the case of unit weights, i.e., for the problem of minimizing the number of late jobs, the

* Accepted by Alexander H. G. Rinnooy Kan; received July 1986. This paper has been with the authors 7 months for 1 revision.

0025-1909/88/3407/0843$01.25

$O(n \log n)$ algorithm of Moore (1968) (sometimes known as Hodgson's algorithm) solves the problem. Moore's algorithm is generalized by Lawler (1976) to the case of agreeable weights (i.e., $p_i < p_j$ implies $w_i \geq w_j$). Lawler and Moore notice that, when all due dates are equal, the problem is identical to the zero-one knapsack problem. Karp (1972) uses this relationship to show that the problem is (binary) NP-hard. Consequently, it is highly unlikely that the problem is polynomially solvable.

Since branch and bound algorithms have been fairly successful in solving knapsack problems (see Martello and Toth 1979 for a review and computational comparison of algorithms), we attempt to generalize their main features to our weighted number of late jobs problem. In particular, we show how the reduction procedure of Ingargiola and Korsh (1973) and the bounding procedure of Dantzig (1957) can be adapted to the weighted number of late jobs problem. In §2 the dominance theorem of Villarreal and Bulfin is given and our reduction procedure is derived. §3 describes the dynamic programming algorithm of Lawler and Moore and derives a lower bound that uses this dynamic programming algorithm to solve a scaled version of the original problem. Two further lower bounding schemes are described in §4: the first is based on Moore's algorithm while the second efficiently solves a linear programming problem obtained by relaxing the integrality constraints of a zero-one programming formulation of the problem. Complete descriptions of branch and bound algorithms which use the three lower bounding schemes are given in §5. §6 reports on computational experience with the dynamic programming and branch and bound algorithms and some concluding remarks are given in §7.

2. Dominance and Reduction

In this section we first present the dominance theorem of Villarreal and Bulfin which is used to eliminate nodes of a search tree in a branch and bound algorithm. The dominance theorem is also used in a reduction algorithm which is described next. The reduction algorithm indicates that some jobs are necessarily early or late; a method is given whereby the due dates of remaining jobs are adjusted to account for these early and late jobs which can then be eliminated from the problem.

The result proved by Villarreal and Bulfin is as follows.

THEOREM 1 (Dominance Theorem). *If, for two jobs i and j, the conditions*

$$p_i \leq p_j, \qquad w_i \geq w_j, \qquad d_i - p_i \geq d_j - p_j \tag{1}$$

hold, then there exists an optimal schedule in which either job i is early or job j is late.

To avoid complications that arise if each condition of (1) is satisfied as an equality, we assume that no two jobs have identical data. If necessary, this can be achieved by perturbing the weights.

If jobs i and j satisfying conditions (1) are found, then the Dominance Theorem is used as follows. If job i is known to be late, then only schedules in which job j is late are considered, and if job j is known to be early then only schedules in which job i is early are considered. Let E_j $(j = 1, \ldots, n)$ denote the set of jobs, including job j, that are early if job j is early and let L_j $(j = 1, \ldots, n)$ denote the set of jobs, including job j, that are late if job j is late.

The details of implementation of the Dominance Theorem for reducing the size of the search tree in a branch and bound algorithm are discussed later. We now concentrate on its use in our reduction algorithm.

Suppose that a heuristic method has been applied to yield feasible sets of early and late jobs. Let UB denote the weighted number of late jobs for this schedule. Clearly, UB is an upper bound on the value of an optimal schedule. Thus, it is required to search for

a schedule with value less than UB. Suppose also that a lower bounding scheme is available which computes a lower bound LB (E, L) on the value of any schedule that is constrained to have a subset E of early jobs and a subset L of late jobs. The exact choice of a heuristic method and of a lower bounding scheme is discussed later.

First we describe an *earliness test* to ascertain whether some job j is necessarily early in the search for a schedule with value less than UB. A lower bound $\text{LB}(\varnothing, L_j)$ on the value of the schedule in which job j (together with the other jobs of L_j found from the Dominance Theorem) is constrained to be late is computed. If $\text{LB}(\varnothing, L_j) \geqslant \text{UB}$, then only schedules in which job j is early are considered. A *lateness test* is defined similarly. If $\text{LB}(E_j, \varnothing) \geqslant \text{UB}$ (where we define $\text{LB}(E_j, \varnothing) = \infty$ if E_j is not a feasible set of early jobs), then only schedules in which job j is late are considered. Depending upon the lower bounding procedure that is adopted, the lateness test may be improved by finding the subset E_j^L of jobs of $\{1, \ldots, n\} - E_j$ that are necessarily late when the jobs of E_j are constrained to be early. It is possible to test in turn each $i \in \{1, \ldots, n\} - E_j$ for membership of E_j^L: $i \in E_j^L$ if and only if $E_j \cup \{i\}$ is not a feasible set of early jobs. However, a more efficient construction for E_j^L is described later. Then if $\text{LB}(E_j, E_j^L) \geqslant \text{UB}$, only schedules in which job j is late are considered. These tests are of the same type as those proposed by Ingargiola and Korsh for the zero-one knapsack problem. They are, however, strengthened by the use of the Dominance Theorem that is not present in the tests of Ingargiola and Korsh.

When either of the above tests are successful the following *reduction procedure*, which removes job j from the problem, is executed. If the earliness test on job j is successful, due dates are reset using

$$\begin{aligned} d_i &= \min\{d_i, d_j - p_j\} \quad \text{for} \quad i = 1, \ldots, j-1, \\ d_i &= d_i - p_j \quad \text{for} \quad i = j+1, \ldots, n, \end{aligned} \tag{2}$$

and job j is removed from the problem. Alternatively, if the lateness test on job j is successful, job j is removed from the problem and a contribution w_j is added to the weighted number of late jobs.

The following result proves the validity of the complete reduction procedure.

THEOREM 2 (Reduction Theorem). *When the earliness or lateness tests are successful, the search for an optimal schedule may be restricted to an* $(n-1)$*-job problem defined by the reduction procedure.*

PROOF. Suppose the earliness test on job j is successful. Clearly a schedule that is consistent with the Dominance Theorem which has value less than UB cannot be found if j is late. We now show that the reduced $(n-1)$-job problem and the original n-job problem with job j constrained to be early are equivalent. Let E be a feasible set of early jobs for the original problem. We now show that $E - \{j\}$ is a feasible set of early jobs for the reduced problem. Recall that the jobs of both E and $E - \{j\}$ are sequenced in EDD order. Let C_i and d_i be the completion time and due date of job i $(i = 1, \ldots, n)$ in the original problem and let C_i^R and d_i^R be the corresponding values for job i $(i = 1, \ldots, n;\ i \neq j)$ in the reduced problem. Let $i \in E$ where $i < j$. Then $C_i \leqslant d_i$ and $C_i \leqslant C_j - p_j \leqslant d_j - p_j$. Thus, $C_i^R = C_i \leqslant \min\{d_i, d_j - p_j\} = d_i^R$ as required. Alternatively, if $i \in E$ with $i > j$, then $C_i^R = C_i - p_j \leqslant d_i - p_j = d_i^R$ as required. This proves that if E is a feasible set of early jobs for the original problem, then $E - \{j\}$ is a feasible set of early jobs for the reduced problem. Conversely, let E be a feasible set of early jobs for the reduced problem. We show that $E \cup \{j\}$ is a feasible set of early jobs for the original problem. Let $i \in E$ where $i < j$. Then $C_i = C_i^R \leqslant d_i^R \leqslant d_i$. Alternatively, let $i \in E$ where $i > j$. Then $C_i = C_i^R + p_j \leqslant d_i^R + p_j = d_i$ as required. Lastly we show that job j is early. Choose the job i sequenced immediately before job j in the original problem. (If

job j is sequenced first, then trivially $C_j = p_j \leqslant d_j$.) Since $C_i^R \leqslant d_i^R \leqslant d_j - p_j$ we have $C_j = C_i^R + p_j \leqslant d_i^R + p_j \leqslant d_j$ as required. Thus, there is a one-to-one correspondence between feasible sets of early jobs for the original and reduced problems which therefore implies that the two problems are equivalent.

If the lateness test on job j is successful, then clearly a schedule that is consistent with the Dominance Theorem which has value less than UB cannot be found if job j is early. Thus, its contribution w_j is added to the weighted number of late jobs after which it is discarded to give a reduced problem which is otherwise equivalent to the original problem. QED

When job j is early, the resetting of due dates according to (2) requires $O(n)$ time. If a set E of jobs are early, then by considering jobs in nonincreasing order of due dates, the reduction procedure can still be implemented on $O(n)$ time as follows. Firstly, the jobs of E are scheduled as late as possible subject to deadlines on their completion times given by due dates. To construct this schedule, the job of E with the largest index is scheduled to be completed at its due date, the job with the second largest index is scheduled to be completed either at its due date or at the start time of the previously scheduled job if this is smaller, etc. Let C_j be the completion time of job j ($j \in E$) in this schedule. Now consider any job i, where $i \in \{1, \ldots, n\} - E$, and suppose that E is partitioned into sets $E_1 = \{j \mid j \in E, j < i\}$ and $E_2 = \{j \mid j \in E, j > i\}$. Then d_i is reset using $d_i = d_i - \sum_{j \in E_1} p_j$ if $E_2 = \varnothing$ or using $d_i = \min\{d_i, C_k - p_k\} - \sum_{j \in E_1} p_j$, where k is chosen so that $k \in E_2$ and k is as small as possible, if $E_2 \neq \varnothing$.

In the reduction algorithm described below, the earliness tests are performed first followed by the lateness tests. The order in which each set of tests is performed is chosen so that the earlier tests are more likely to be successful than the later ones. In the algorithm, E and L are the set of jobs that must be early and late respectively and S is the set of remaining jobs. Also, W is the total weight of jobs of L.

Reduction Algorithm

Step 1. Compute LB($\varnothing$, $\varnothing$) and apply a heuristic method to find UB. If LB($\varnothing$, $\varnothing$) = UB, stop. Otherwise find an ordering $(\sigma(1), \ldots, \sigma(n))$ such that $w_{\sigma(1)}/p_{\sigma(1)} \geqslant \cdots \geqslant w_{\sigma(n)}/p_{\sigma(n)}$, set $S = \{1, \ldots, n\}$, $E = \varnothing$, $L = \varnothing$, $W = 0$, $j = 1$ and $k = n$.

Step 2. If $\sigma(j) \notin S$ go to Step 4. Otherwise find the set $L_{\sigma(j)} \cap S$ and, considering only the jobs of S, compute LB($\varnothing$, $L_{\sigma(j)} \cap S$). If LB($\varnothing$, $L_{\sigma(j)} \cap S$) + $W <$ UB go to Step 4.

Step 3. Set $S = S - \{\sigma(j)\}$, $E = E \cup \{\sigma(j)\}$, $d_i = \min\{d_i, d_{\sigma(j)} - p_{\sigma(j)}\}$ for $i \in S$ and $i < \sigma(j)$, and $d_i = d_i - p_{\sigma(j)}$ for $i \in S$ and $i > \sigma(j)$. For each $i \in S$ with $p_i > d_i$, set $S = S - \{i\}$, $L = L \cup \{i\}$ and $W = W + w_i$.

Step 4. If $j = n$, go to Step 5. Otherwise set $j = j + 1$ and go to Step 2.

Step 5. If $\sigma(k) \notin S$ go to Step 7. Otherwise find sets $E_{\sigma(k)} \cap S$ and $E^L_{\sigma(k)} \cap S$ and, considering only the jobs of S, compute LB($E_{\sigma(k)} \cap S$, $E^L_{\sigma(k)} \cap S$). If LB($E_{\sigma(k)} \cap S$, $E^L_{\sigma(k)} \cap S$) + $W <$ UB, go to Step 7.

Step 6. Set $S = S - \{\sigma(k)\}$, $L = L \cup \{\sigma(k)\}$ and $W = W + w_{\sigma(k)}$.

Step 7. If $k = 1$, stop. Otherwise set $k = k - 1$ and go to Step 5.

In the reduction algorithm, the value LB($\varnothing$, $L_{\sigma(j)} \cap S$) that is computed in Step 2 is obtained by finding a lower bound for the problem with jobs $S - L_{\sigma(j)}$ and adding the contribution $\sum_{i \in L_{\sigma(j)} \cap S} w_i$. To obtain the value LB($E_{\sigma(k)} \cap S$, $E^L_{\sigma(k)} \cap S$) that is computed in Step 5, the reduction procedure is first used to reset due dates to account for the early jobs $E_{\sigma(k)} \cap S$. The set $E^L_{\sigma(k)} \cap S$ is given by $E^L_{\sigma(k)} \cap S = \{i \mid i \in S, p_i > d_i\}$ for the reset due dates d_i. A lower bound for the problem with jobs $S - E_{\sigma(k)} - E^L_{\sigma(k)}$ having reset due dates is added to the contribution $\sum_{i \in E^L_{\sigma(k)} \cap S} w_i$ to give the required lower bound.

There may be an upper bounding procedure implicit in the lower bounding procedure that is used in the algorithm. If this is the case, then in Step 2 and in Step 5 where

lower bounds are computed, an upper bound is computed. If the value of the newly computed upper bound is less than UB, then UB is updated accordingly.

The computational complexity of the reduction algorithm depends upon the heuristic and the lower bounding procedure employed. The heuristic in Step 1 is applied once and the lower bounding procedure is applied at most $2n + 1$ times. The ordering of Step 1 requires $O(n \log n)$ time. Excluding the computation of the lower bounds, all other steps are executed at most n times with each execution requiring at most $O(n)$ time.

3. Dynamic Programming

The Dynamic Programming Algorithm

Lawler and Moore's pseudopolynomial dynamic programming algorithm, which is denoted by algorithm DPLM, is described in this section. Let $T = \min \{d_n, \sum_{j=1}^{n} p_j\}$ be the maximum completion of jobs sequenced early. Also, let $f_j(t)$ ($j = 1, \ldots, n$; $t = 0, \ldots, T$) be the weighted number of late jobs when jobs $1, \ldots, j$ are scheduled so that the last early job is completed at time t. Then the recursion is

$$f_j(t) = \min \{f_{j-1}(t) + w_j, f_{j-1}(t - p_j) + M \max \{t - d_j, 0\}\} \tag{3}$$

where $f_0(0) = 0, f_0(t) = \infty$ for $t = 1, \ldots, T, f_j(t) = \infty$ for $j = 0, \ldots, n$ and $t < 0$, and M is a large number ($M > \sum_{j=1}^{n} w_j$). The case that $f_j(t) = f_{j-1}(t) + w_j$ corresponds to the decision that job j is late and the case that $f_j(t) = f_{j-1}(t - p_j)$ with $t \leqslant d_j$ corresponds to the decision that job j is early. Equation (3) is used to compute $f_j(t)$ for $j = 1, \ldots, n$ and $t = 0, \ldots, T$ after which the minimum weighted number of late jobs is given by $\min_{t=0,\ldots,T} \{f_n(t)\}$. Clearly, algorithm DPLM requires $O(nT)$ time.

As is the case with most dynamic programming algorithms, the applicability of algorithm DPLM is limited by its storage requirements. We discuss first the problem of finding the weighted number of late jobs without determining the corresponding partition into early and late jobs. Let $p_{\max}$ denote the maximum processing time. During the computation of $f_j(t)$ and in subsequent computations, the values $f_0(t), \ldots, f_{j-2}(t)$ for $t = 0, \ldots, T$ and, for the case $t > p_j$, the values $f_{j-1}(0), \ldots, f_{j-1}(t - p_j - 1)$ are not required. Thus, for $t \geqslant p_j$ only the previously computed values $f_{j-1}(t - p_j), \ldots, f_{j-1}(T), f_j(0), \ldots, f_j(t - 1)$ are stored and for $t < p_j$ only the previously computed values $f_{j-1}(0), \ldots, f_{j-1}(T), f_j(0), \ldots, f_j(t - 1)$ are stored. In either case, at most $T + p_j + 1$ values need to be stored. Therefore, at any stage at most $T + p_{\max} + 1$ values need to be stored.

We return to the problem of finding the optimal set of early and late jobs. One possibility is to use backup storage to store the values $f_j(t)$ for $j = 1, \ldots, n$ and $t = 0, \ldots, T$. Then a simple backtracing procedure generates the sets of early and late jobs. On the other hand, if only core storage is available, we proceed as follows. Corresponding to each $f_j(t)$ we define the zero-one variables $\delta_j(t)$, where

$$\delta_j(t) = \begin{cases} 1 & \text{if} \quad f_j(t) = f_{j-1}(t) + w_j, \\ 0 & \text{if} \quad f_j(t) = f_{j-1}(t - p_j) + M \max \{t - d_j, 0\}, \end{cases}$$

i.e., in the computation of $f_j(t)$, $\delta_j(t)$ takes the value 1 if job j is late and takes the value 0 otherwise. The decisions $\delta_1(t), \ldots, \delta_n(t)$ corresponding to the values $f_1(t), \ldots, f_n(t)$ may be stored as a string variable with value $d(t) = \sum_{j=1}^{n} 2^{n-j}\delta_j(t)$. The decision values $\delta_j(t)$ ($j = 1, \ldots, n$) can easily be decoded from $d(t)$ ($t = 0, \ldots, T$) thus allowing backtracing to generate the optimal set of early and late jobs. In computing $d(t)$ it is assumed that $n \leqslant W$, where W is the maximum number of bits used in each word. If $n > W$, $\lceil n/W \rceil$ words are needed for each t value to represent the string of decisions.

To summarize, the storage space required to perform the recursion in Algorithm DPLM is $T + p_{\max} + 1$ words. The method of storing decision variables described above requires an additional $\lceil n/W \rceil (T + 1)$ words of storage.

In our implementation, the available core storage for performing the recursion and storing the decision variables is 46000 words and $W = 50$. Thus, algorithm DPLM is capable of storing problems for which $p_{\max} + (1 + \lceil n/50 \rceil)(T + 1) \leqslant 46000$. When $p_{\max} + T + 1 \leqslant 46000$ but $p_{\max} + T + 1 > 46000 - \lceil n/50 \rceil(T + 1)$, the recursion is performed to find the weighted number of late jobs, although the problem is regarded as unsolved because an optimal schedule is not obtained.

A Lower Bound from Dynamic Programming

If $T + p_{\max} + 1$ words of storage are not available to perform the recursion, this dynamic programming approach can still be used to provide a lower bound on the weighted number of late jobs. We define a scaled problem for which processing times and due dates are given by $p_j^S = \lfloor p_j/K \rfloor$ and $d_j^S = \lfloor d_j/K \rfloor$ $(j = 1, \ldots, n)$ where the *scaling factor* is $1/K$ $(K \geqslant 1)$. The following result shows the solution of the scaled problem (obtained using Algorithm DPLM) provides a lower bound.

THEOREM 3. *The solution of the scaled problem is a lower bound for the solution of the original problem.*

PROOF. We show that any feasible set E of early jobs for the original problem is also a feasible set of early jobs for the scaled problem. Let C_j and C_j^S denote the completion time of job j $(j \in E)$ in the original and scaled problems when the jobs of E are sequenced in EDD order. Consider any early job j $(j \in E)$ and let $J = \{i \mid C_i \leqslant C_j\}$ be the set of early jobs not completed after job j. Then $C_j = \sum_{i \in J} p_i \leqslant d_j$. We have $C_j^S = \sum_{i \in J} p_i^S \leqslant \sum_{i \in J} p_i/K \leqslant d_j/K$. Since $\sum_{i \in J} p_i^S \leqslant d_j/K$ and $\sum_{i \in J} p_i^S$ is integer, $\sum_{i \in J} p_i^S \leqslant \lfloor d_j/K \rfloor$ giving $C_j^S \leqslant d_j^S$ as required. QED

The scaling factor is chosen so that the maximum number $\lfloor (T + p_{\max} + 1)/K \rfloor$ of words of storage required by algorithm DPLM is available. It is generally expected that this lower bound becomes less tight as K increases. We denote this lower bound by LBDP. It requires $O(n)$ time since the value of T/K is bounded above by the number of words of available storage which is fixed. The original problem can be solved by using LBDP as a lower bound in a branch and bound algorithm.

4. Lower Bounds

Consider the following zero-one programming formulation of the problem. The variables u_j $(j = 1, \ldots, n)$ are defined as

$$u_j = \begin{cases} 1 & \text{if job } j \text{ is late;} \\ 0 & \text{if job } j \text{ is early.} \end{cases}$$

The problem becomes

$$\text{(P)} \qquad \text{minimize} \quad \sum_{j=1}^{n} w_j u_j$$

$$\text{subject to} \quad \sum_{i=1}^{j} p_i(1 - u_i) \leqslant d_j \qquad (j = 1, \ldots, n), \tag{4}$$

$$u_j \in \{0, 1\} \qquad (j = 1, \ldots, n). \tag{5}$$

Note that when $d_1 = \cdots = d_n$, the first $n - 1$ constraints of (4) are redundant. Substituting $u_j = 1 - x_j$ $(j = 1, \ldots, n)$ in (P) leads to the zero-one knapsack problem formulated in the usual way.

A Lower Bound from the Number of Late Jobs

Let n_j^L $(j = 1, \ldots, n)$ denote the minimum number of late jobs for the problem containing jobs $1, \ldots, j$. Clearly, $n_j^E = j - n_j^L$ is the maximum number of early jobs for

this problem. The values n_j^L are generated as a by-product of Moore's algorithm and are implicit in (4) and (5). Clearly, the constraints

$$\sum_{i=1}^{j} (1 - u_i) \leqslant n_j^E \qquad (j = 1, \ldots, n) \tag{6}$$

are a valid set of inequalities for the weighted number of late jobs problem and may be added to (P) without altering its solution. However, for our first lower bound we replace constraints (4) with constraints (6) to give the problem

$$(\mathrm{P}^{\mathrm{NLJ}}) \qquad \text{minimize} \quad \sum_{j=1}^{n} w_j u_j$$
$$\text{subject to (5) and (6).}$$

Comparing problem ($\mathrm{P}^{\mathrm{NLJ}}$) with problem (P), we observe that ($\mathrm{P}^{\mathrm{NLJ}}$) is also a weighted number of late jobs problem: for ($\mathrm{P}^{\mathrm{NLJ}}$) job j ($j = 1, \ldots, n$) has a unit processing time and has a due date n_j^E. Consequently, problem ($\mathrm{P}^{\mathrm{NLJ}}$) is solved using the following version of Moore's algorithm which is modified to handle agreeable weights. Initially, set $u_j = 0$ for $j = 1, \ldots, n$, set $S = \{1, \ldots, n\}$ and let j denote the index of the first constraint of (6) that is violated. Choose i such that $w_i = \min_{h \in \{1, \ldots, j\} \cap S} \{w_h\}$, set $S = S - \{i\}$ and set $u_i = 1$. This procedure is repeated until values u_j ($j = 1, \ldots, n$) are obtained which satisfy each constraint of (6). Let LBNLJ denote the value of the lower bound obtained from the solution of problem ($\mathrm{P}^{\mathrm{NLJ}}$). The computation of LBNLJ requires $O(n \log n)$ time.

The lower bound LBNLJ is similar to the lower bounds derived by Villarreal and Bulfin. In both of their lower bounds they effectively solve a weighted number of late jobs problem with unit processing times. However, their due dates for this problem in both cases are not, in general, as low as our values n_j^E ($j = 1, \ldots, n$) which are the best possible. Thus, LBNLJ dominates both bounds of Villarreal and Bulfin.

A Lower Bound from the Linear Programming Relaxation

Our last lower bound is obtained from the solution of the linear programming problem in which the integrality constraints in problem (P) are relaxed. More precisely, constraints (5) are replaced by

$$0 \leqslant u_j \leqslant 1 \qquad (j = 1, \ldots, n) \tag{7}$$

and our lower bound is the solution of the linear programming problem

$$(\mathrm{LP}) \qquad \text{minimize} \quad \sum_{j=1}^{n} w_j u_j$$
$$\text{subject to (4) and (7).}$$

We now describe an efficient algorithm to solve problem (LP). The algorithm successively considers a sequence of linear programming problems ($\mathrm{LP}^{(n)}$), ..., ($\mathrm{LP}^{(1)}$), where problem ($\mathrm{LP}^{(k)}$) is defined by

$$(\mathrm{LP}^{(k)}) \qquad \text{minimize} \quad \sum_{j=1}^{n} w_j u_j$$
$$\text{subject to} \quad \sum_{i=1}^{j} p_i(1 - u_i) \leqslant d_j \qquad (j = 1, \ldots, k-1), \tag{8}$$
$$\sum_{j=1}^{n} p_j(u_j^{(k+1)} - u_j) \leqslant d_k, \tag{9}$$

$$0 \leqslant u_j \leqslant u_j^{(k+1)} \qquad (j = 1, \ldots, n). \tag{10}$$

The upper bound values $u_j^{(k+1)}$ $(j = 1, \ldots, n)$ are determined by the algorithm whilst considering the problem $(\mathrm{LP}^{(k+1)})$, where initially $u_j^{(n+1)} = 1$ $(j = 1, \ldots, n)$. Clearly, problem $(\mathrm{LP}^{(n)})$ is identical to our relaxed problem (LP).

Roughly speaking, when considering problem $(\mathrm{LP}^{(k)})$, the algorithm reduces the upper bound values $u_j^{(k+1)}$ on u_j for $j = k, \ldots, n$ to lower values $u_j^{(k)}$. If possible, the reduction continues until $\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u_j^{(k)}) = d_k - d_{k-1}$. When there is a choice, a value corresponding to a job i having w_i/p_i as large as possible is selected for reduction. Problem $(\mathrm{LP}^{(1)})$ is equivalent to the linear programming relaxation of the zero-one knapsack problem and is solved by the well-known procedure of Dantzig.

In the formal statement of the algorithm below, the index k shows which problem $(\mathrm{LP}^{(k)})$ is under consideration, the value $d^{(k)}$ provides a measure of the total decrease in $\sum_{j=1}^{n} p_j u_j^{(k)}$ that should ideally be achieved during iteration k and S is the set of jobs from which the decrease in their upper bound constraints is chosen.

Algorithm LP

Step 1. Set $u_j^{(n+1)} = 1$ for $j = 1, \ldots, n$, set $S = \varnothing$, set LBLP $= \sum_{j=1}^{n} w_j$, set $d_0 = 0$ and set $k = n$.

Step 2. Set $S = S \cup \{k\}$, set $d^{(k)} = d_k - d_{k-1}$ and set $u_j^{(k)} = u_j^{(k+1)}$ for $j = 1, \ldots, n$.

Step 3. Select a job i, where $i \in S$, so that w_i/p_i is as large as possible. If $p_i u_i^{(k)} < d^{(k)}$, go to Step 4; otherwise go to Step 5.

Step 4. Set $d^{(k)} = d^{(k)} - p_i u_i^{(k)}$, set LBLP $=$ LBLP $- w_i u_i^{(k)}$, set $u_i^{(k)} = 0$ and set $S = S - \{i\}$. If $S = \varnothing$, go to Step 6; otherwise go to Step 3.

Step 5. Set LBLP $=$ LBLP $- w_i d^{(k)}/p_i$ and set $u_i^{(k)} = u_i^{(k)} - d^{(k)}/p_i$.

Step 6. If $k = 1$, then stop with LBLP as the lower bound and $u_j = u_j^{(1)}$ $(j = 1, \ldots, n)$ as the corresponding values of the variables. Otherwise set $k = k - 1$ and go to Step 2.

It should be noted that the superscripts relating to the index k that are used in Algorithm LP can be dropped without affecting the solution that is generated. The superscripts are used solely to simplify the explanation in the proof that the algorithm generates an optimal solution to problem (LP). This proof is given in the Appendix.

THEOREM 4. *Algorithm LP generates an optimal solution to problem (LP).*

The computational complexity of Algorithm LP is discussed now. We first ascertain that each step of the algorithm is executed at most $O(n)$ times. Clearly, Step 1 is an initialization step that is performed once only, while Steps 2 and 6 are both executed once during each of the n iterations of the algorithm. Step 3 is executed immediately after Step 2 during each iteration of the algorithm and, possibly, after each execution of Step 4. However, Step 4 is executed at most n times since the cardinality of S is reduced by one each time. Thus, Step 3 is executed at most $2n$ times and Step 4 is executed at most n times. Step 5 signals an end to the current iteration and, therefore, is executed at most n times. We have now verified that no step of the algorithm is executed more than $O(n)$ times. We describe next how the algorithm can be implemented so that each step, with the exception of Step 1, requires at most $O(\log n)$ time. By ordering the jobs of S according to the ratios w_i/p_i, an insertion required in Step 2, a deletion required in Step 4 and the selection of the job with the largest ratio required in Step 3 can each be performed in $O(\log n)$ time. All other statements require constant time. We have now established that each of the $O(n)$ executions of Steps 2 to 6 requires at most $O(\log n)$ time. Step 1 requires $O(n)$ time but is executed once only. Therefore, the overall time requirement for the algorithm is $O(n \log n)$.

An algorithm to solve a generalized version of problem (LP) in $O(n^2)$ time is proposed by Faaland (1984). This algorithm also considers a sequence of n linear programming

problems but proceeds in the reverse direction to Algorithm LP by setting new lower bounds on $u_1, \ldots, u_k$ at iteration k ($k = 1, \ldots, n$). Our 'backward' approach yields a more efficient algorithm than the 'forward' approach of Faaland for problem (LP).

Having applied the algorithm, the solution generated may be used to obtain an upper bound. Let $E = \{j \mid j \in \{1, \ldots, n\}; u_j^{(1)} = 0\}$. Clearly, E forms a feasible set of early jobs. The value of the upper bound is the total weight for the other jobs, i.e., the upper bound is $\sum_{j \in L} w_j$, where $L = \{1, \ldots, n\} - E$.

5. The Algorithms

Implementation of the Reduction Algorithm

Recall that the Reduction Algorithm as described in §2 requires the use of a lower bounding procedure and an upper bound. Any of the lower bounding schemes LBDP, LBNLJ or LBLP may be selected. However, initial experiments show that LBLP produces tight lower bounds without requiring excessive computation. It is thus selected for use.

In Step 1 of the Reduction Algorithm, Algorithm LP is applied to the original problem to obtain $u^{(1)} = (u_1^{(1)}, \ldots, u_n^{(1)})$ and LBLP. The following heuristic which uses $u^{(1)}$ is then applied. Firstly, feasible sets of early and late jobs $E^H = \{j \mid j \in \{1, \ldots, n\}, u_j^{(1)} = 0\}$ and $L^H = \{1, \ldots, n\} - E^H$ are constructed. In an attempt to improve this solution, for any $j \in L^H$, if $E^H \cup \{j\}$ is a feasible set of early jobs, then set $E^H = E^H \cup \{j\}$ and $L^H = L^H - \{j\}$. There are $|L^H|$ such tests that are performed by choosing $j \in L^H$ in nonincreasing order of w_j/p_j. When no further jobs can be transferred from L^H to E^H without causing infeasibility, the upper bound $\mathrm{UB} = \sum_{j \in L^H} w_j$ is computed.

The computational complexity of the Reduction Algorithm can now be derived. The heuristic is computed from $u^{(1)}$ by performing at most n tests each of which requires $O(n)$ time. Thus, it requires $O(n^2)$ time. Since Algorithm LP requires $O(n \log n)$ time, the $2n + 1$ applications of it in the Reduction Algorithm require $O(n^2 \log n)$ time. Lastly, the other steps of the Reduction Algorithm require $O(n^2)$ time. Therefore, the computational complexity of the Reduction Algorithm is dominated by the computation of the lower bounds that requires $O(n^2 \log n)$ time.

To reduce computation, certain tests are eliminated from the Reduction Algorithm if they are unlikely to be successful. For each job $\sigma(j)$ yielding a value $u_{\sigma(j)}^{(1)} = 1$ in Step 1, the earliness test on job $\sigma(j)$ in Step 2 is eliminated. Similarly, for each job $\sigma(j)$ yielding $u_{\sigma(j)}^{(1)} = 0$ in Step 1, the lateness test on job $\sigma(j)$ in Step 5 is not executed.

Branch and Bound Algorithms

We describe next a branch and bound algorithm that can incorporate any of the lower bounding schemes LBDP, LBNLJ or LBLP. In our branch and bound search tree, corresponding to each node is a set E of jobs that must be early, a set L of jobs that must be late and a set F of free jobs that have not been fixed. A binary branching rule selects job j from F with w_j/p_j as large as possible and constructs two descendant nodes in which job j is constrained to be late and early respectively. For the first descendant in which job j is late, the set L_j of jobs that, according to the Dominance Theorem, are additionally late when job j is late is computed and the sets of late and free jobs are updated from the corresponding sets at the parent node using $L = L \cup L_j$ and $F = F - L_j$ (with the set E remaining the same as at the parent node). For the second descendant in which job j is constrained to be early, the sets of early and free jobs are updated from the corresponding sets at the parent node using $E = E \cup \{j\}$ and $F = F - \{j\}$ (with the set L remaining the same as at the parent node). Furthermore, for each $k \in F$ such that $E \cup \{k\}$ is an infeasible set of early jobs, the set of late and free jobs are reset using $L = L \cup \{k\}$ and $F = F - \{k\}$. Note that the Dominance Theorem is not used in fixing

jobs at this second descendant since, due to the way in which job j is selected, the conditions are unlikely to be satisfied.

A newest active node search is used in the branch and bound algorithm. Thus, a node in which job j is constrained to be early is always searched before the corresponding node in which job j is constrained to be late. The use of this search strategy economizes on the storage requirements of the algorithm: the maximum number of nodes stored at any time is n.

A special feature employed to increase the efficiency of our branch and bound algorithm is the elimination of lower bounding calculations at certain nodes of the search tree. Suppose that the lower bounding scheme LBLP is employed and that for any node of the search tree, having E, L and F as sets of early, late and free jobs, Algorithm LP generates values $u^{(1)}$ of the variables. Let job j be selected by the branching rule at this node. For the descendant node in which job j is constrained to be early, if $u_j^{(1)} = 0$, then the lower bound is often identical with that of its parent and is not recomputed. An analogous elimination of lower bounding calculations is also used when LBDP or LBNLJ is employed. In other cases where a lower bound is required, the reduction procedure is first applied to account for the sets of early and late jobs at that node after which the relevant lower bound is computed.

The branch and bound algorithms that employ the lower bounds LBDP, LBNLJ and LBLP are denoted by BBDP, BBNLJ and BBLP respectively.

6. Computational Experience

Our first set of experiments was designed to test the relative effectiveness of the algorithms DPLM, BBDP, BBNLJ and BBLP and the corresponding algorithms R + DPLM, R + BBDP, R + BBNLJ and R + BBLP in which the Reduction Algorithm is applied prior to dynamic programming or branch and bound. The algorithms were tested on problems with 50, 100, 150 and 200 jobs that were generated as follows. For each job j an integer processing time p_j and an integer weight w_j were generated from the uniform distribution [1, 100]. The 'hardness' of a problem is likely to depend on the values of the due dates relative to the processing times. Two parameters d^l and d^u were chosen to provide lower and upper bounds on the relative values of the due dates. Having selected d^l and d^u and having computed $P = \sum_{i=1}^{n} p_i$, an integer due date d_j was generated from the uniform distribution $[Pd^l, Pd^u]$ for each job j. For each value of n, four problems were generated for each of the 14 pairs of values d^l and d^u, where $d^l \in \{0.2, 0.4, 0.6, 0.8\}$, $d^u \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and $d^l \leq d^u$. This yields 56 problems for each value of n of which the 16 with $d^l = d^u$ are knapsack problems.

Each of the 8 algorithms was coded in FORTRAN 5 and run on a CDC 7600 computer. Storage limits for the dynamic programming algorithms DPLM and R + DPLM are specified in §3. To assess the performance of algorithms BBDP and R + BBDP, the scaling factor was chosen so that the recursion could be performed using $1000 + p_{\max}$ words of core storage. Initial experiments indicate that these algorithms perform best when the scaling factor is as close to 1 as possible. Nevertheless, our choice of scaling factor gives a general indication of the effectiveness of the lower bound LBDP in a branch and bound algorithm. A time limit of 60 seconds was placed on the six algorithms that employ branch and bound. Whenever a problem was not solved within the time limit, computation was abandoned for that problem.

Comparative computational results for the 8 algorithms are given in Table 1 which lists average computation times, numbers of unsolved problems, average numbers of recursion equations per job for the dynamic programming algorithms and average numbers of nodes for the branch and bound algorithms. These results show BBLP to be the most effective of the algorithms that do not employ the Reduction Algorithm. For

TABLE 1
Computational Results for All Algorithms

	Algorithm	Value of n			
		50	100	150	200
ACT:NU	DPLM	0.29	1.18	2.63	4.40:15
	R + DPLM	0.05	0.19	0.38	0.65
	BBDP	2.11	20.80:5	—	—
	R + BBDP	0.05	0.32	1.62:1	2.96:2
	BBNLJ	1.29	15.61:11	—	—
	R + BBNLJ	0.11	1.26:1	3.67:3	5.17:4
	BBLP	0.05	0.22	0.93	2.22:1
	R + BBLP	0.05	0.20	0.52	1.58
ANR/ANN	DPLM	1747	3635	5420	7155
	R + DPLM	132	167	257	306
	BBDP	36	175	—	—
	R + BBDP	1	2	10	18
	BBNLJ	2463	16079	—	—
	R + BBNLJ	257	2549	5822	7912
	BBLP	45	105	425	807
	R + BBLP	11	32	255	778

ACT: average computation time in seconds (or a lower bound on the average when there are unsolved problems); NU: number of unsolved problems if any; ANR: average number of recursion equations per job to be solved, i.e., the average value of T; ANN: average number of nodes (or a lower bound on the average when there are unsolved problems).

algorithm DPLM the number of recursion equations that are solved is too large to make it competitive with BBLP. For BBDP and BBNLJ the lower bounds are respectively too time consuming and too weak to yield a competitive algorithm.

It is also clear from Table 1 that algorithms R + DPLM, R + BBDP and R + BBNLJ are vastly superior to their counterparts in which the Reduction Algorithm is not applied. Additionally, the superiority of R + BBLP over BBLP is apparent although, in this case, the difference in performance is less marked. Through the use of the Reduction Algorithm, average numbers of jobs are reduced from 50, 100, 150 and 200 to 4, 6, 10 and 11 respectively. For algorithms R + DPLM and R + BBDP, the Reduction Algorithm decreases the number of recursion equations sufficiently to amply compensate for its computational requirements. In comparison to BBNLJ, algorithm R + BBNLJ performs well because the lower bound LBLP that is used in the Reduction Algorithm overcomes the weakness of LBNLJ. The effect of adding the Reduction Algorithm to BBLP is, mainly, to reduce the number of jobs and hence the overheads in the branch and bound algorithm. We conclude that the Reduction Algorithm is a powerful device for reducing problem size and should always be applied prior to the application of a dynamic programming or branch and bound algorithm.

The results of Table 1 show that R + DPLM and R + BBLP are the most efficient algorithms. For our selection of test problems R + DPLM performs better than R + BBLP. Unlike R + BBLP, however, the performance of R + DPLM depends largely on the size of processing times: it would be simple to generate problems with large processing times for which R + BBLP generates a solution much more quickly than R + DPLM. Therefore, it is impossible to make a general conclusion as to which is the better of these two algorithms.

To further investigate algorithms R + DPLM and R + BBLP, problems with 250, 500, 750 and 1000 jobs were generated using the method described above. A time limit

of 500 seconds was used for algorithm R + BBLP. For these larger problems, Table 2 gives average and median computation times, where necessary numbers of unsolved problems, average numbers of recursion equations per job for R + DPLM and average numbers of nodes generated by R + BBLP. The results show that both algorithms are surprisingly successful at solving these large sized problems. A closer examination shows that the Reduction Algorithm substantially reduces problem size: the average and maximum numbers of jobs in the reduced problem are 10, 18, 22, 14, and 38, 116, 218, 271 for the respective n values 250, 500, 750, 1000. Average computation times are less for R + DPLM, although median computation times are similar for both algorithms. This indicates that a small number of difficult problems accounts for the differences in average computation times. The computation time required to execute the Reduction Algorithm is large compared with the time required to perform the dynamic programming recursion in R + DPLM and, for most problems, with the time required by the branch and bound algorithm in R + BBLP.

The success of R + DPLM and R + BBLP in solving large problems is attributed to the ability of the Reduction Algorithm to eliminate jobs from the problem. For the Reduction Algorithm to be successful, the lower bound LBLP that it uses must be powerful. An examination of LBLP, as applied to the original problem, verifies that it is close to the optimum. For instance, when $n = 200$ the average value of OPT − LBLP, where OPT is the value of an optimal solution, is 2.2 which is substantially less than the average job weight of 50.5. The lower bound LBLP is close to the optimum because the optimal solution of problem (LP) has few variables taking non-integer values. This is again demonstrated for $n = 200$ where the solution of problem (LP) has an average of 1.2 and a maximum of 3 variables taking non-integer values. The relaxation of integrality constraints in randomly generated problems appears to produce only a small reduction in the objective function.

Some further tests were performed for problems with 250, 500, 750 and 1000 jobs where processing times and weights were generated from the uniform distribution [1, 10^6]. Such problems with large numbers of distinct processing times and weights are expected to be harder than those generated originally where many processing times and many weights are equal because the conditions of the Dominance Theorem are less likely to be satisfied. Computational results confirm expectations that the Reduction Algorithm is less successful for these new problems with a large range of processing times and weights. Although computation times are larger for these new problems, they do not appear to be substantially harder. The number of problems for which R + BBLP generates over 1000 nodes is about the same in the original and in these further tests.

The influence of parameters d^l and d^u on problem hardness warrants discussion.

TABLE 2

Computational Results for Large Problems

	R + DPLM			R + BBLP		
n	ACT:NU	MCT	ANR	ACT:NU	MCT	ANN
250	0.85	1.06	292	0.85	1.06	25
500	3.43	4.37	501	6.48	4.39	2999
750	7.82	9.95	700	20.11:1	9.90	7449
1000	11.27:1	11.47	424	19.82:1	11.57	2253

ACT: average computation time in seconds (or a lower bound on the average when there are unsolved problems); NU: number of unsolved problems if any; MCT: median computation time in seconds; ANR: average number of recursion equations per job to be solved, i.e., the average value of T; ANN: average number of nodes (or a lower bound on the average when there are unsolved problems).

When $d^l = 0.2$ and $d^u = 0.4$, problems are generated that, after the application of the Reduction Algorithm, have the largest numbers of jobs on average. After $d^l = 0.2$ and $d^u = 0.4$, the next hardest problems are those with $d^l = 0.2$ or $d^l = 0.4$ and $d^u = 0.6$. Problems with $d^u = 1.0$ have very few late jobs and are usually solved very quickly by the Reduction Algorithm alone. The other problem classes, including the knapsack problems with $d^l = d^u$, appear to be of roughly equal difficulty although the computation time required for the Reduction Algorithm, and therefore the overall computation time, tends to increase as due dates increase. The computation time and the storage requirement of DPLM depend on the value of T, where for our problems $T \leq d^u \sum_{i=1}^{n} p_i$. It is fortunate, therefore, that the hardest problem class ($d^l = 0.2$ and $d^u = 0.4$) does not correspond to large d^u.

The results of Martello and Toth for the zero-one knapsack problem indicate that problems with positively correlated processing times and weights are significantly harder than the *uncorrelated* (UC) problems considered in our first experiments. A set of *weakly correlated* (WC) problems and a set of *strongly correlated* (SC) problems with 50, 100, 150 and 200 jobs were generated as follows. The processing times and the due dates were generated using the method described above for the uncorrelated problems. For the weakly correlated problems an integer weight w_j from the uniform distribution $[p_j + 1, p_j + 20]$ was generated for each job j and for the strongly correlated problems an integer weight $w_j = p_j + 20$ was computed for each job j. Thus, 56 weakly correlated and 56 strongly correlated problems were generated for each value of n.

Results for our final experiment in which algorithms R + DPLM and R + BBLP are compared for problem types UC, WC and SC are given in Table 3. Because of the anticipated difficulty in solving the correlated problems, a time limit of 60 seconds was placed on algorithm R + BBLP. It is clear from Table 3 that problem type SC is much harder than type WC which, in turn, is much harder than type UC. This is confirmed by the reduced problem sizes: for the 50, 100, 150 and 200 job problems, the respective average numbers of jobs in the reduced problem are 4, 6, 10 and 11 for type UC, 23, 34,

TABLE 3

The Effect of Correlation Between Processing Times and Weights

	Algorithm	Problem Type	Value of n			
			50	100	150	200
ACT:NU	R + DPLM	UC	0.05	0.19	0.38	0.65
		WC	0.17	0.54	1.07	1.71
		SC	0.26	0.98	1.92	3.17
	R + BBLP	UC	0.05	0.20	0.52	1.58
		WC	0.24	1.75:1	4.12:2	3.71:2
		SC	8.75:6	26.10:21	—	—
ANR/ANN	R+ DPLM	UC	132	167	257	306
		WC	678	1069	1390	1500
		SC	1022	2050	2636	3278
	R + BBLP	UC	11	32	255	778
		WC	230	1129	1910	984
		SC	15115	25749	—	—

ACT: average computation time in seconds (or a lower bound on the average when there are unsolved problems); NU: number of unsolved problems if any; ANR: average number of recursion equations per job to be solved, i.e., the average value of T; ANN: average number of nodes (or a lower bound on the average when there are unsolved problems).

46 and 48 for type WC and 32, 59, 80 and 92 for type SC. Algorithm R + DPLM is able to solve all correlated problems that were generated although with larger computation times than those for type UC: the difference is largely due to the larger numbers of jobs in the reduced problem for the correlated problems. However, for algorithm R + BBLP a total of 5 type WC problems are unsolved after the 60 second time limit. The type SC problems with $n = 100$ proved to be so difficult for R + BBLP that the larger type SC problems were not attempted with this algorithm. The results of Table 3 conclusively show that for hard correlated problems, R + DPLM is the only available algorithm that may possibly solve them.

7. Concluding Remarks

We have developed a powerful new lower bound (LBLP) by deriving an efficient algorithm to solve the linear programming relaxation of the original problem. This lower bound is used in a Reduction Algorithm that indicates when jobs can be eliminated from the problem. The resulting reduced problem can either be solved by Lawler and Moore's dynamic programming algorithm (DPLM) or by a branch and bound algorithm that uses the new lower bound (BBLP), with DPLM having preference unless processing times are very large. Computational results show that due to the success of the Reduction Algorithm in eliminating jobs, large problems can be solved using this approach.

Appendix

PROOF OF THEOREM 4. Initially, some assumptions are made that simplify the argument that follows. Firstly, we assume that $d_1 = 0$. This can be achieved, if necessary, by adding a dummy job with arbitrary positive processing time, with zero weight and with zero due date. The constraint $p_1(1 - u_1) \leqslant d_1$ in (P) and (LP) ensures that $u_1 = 1$ in both problems. Thus, a dummy job does not affect the constraints on the other jobs and does not contribute to the weighted number of late jobs. Secondly, we observe that if a job, other than job 1, has zero weight, then it can be selected to be late without increasing the weighted number of late jobs and can thus be removed from the problem. Thus, we may assume that $w_j > 0$ ($j = 2, \ldots, n$). Thirdly, we assume that each ratio w_j/p_j is distinct: this can easily be achieved by, if necessary, perturbing the weights. Fourthly, we assume that due dates are reset to ensure that $d_j \leqslant d_{j-1} + p_j$ for $j = 2, \ldots, n$. To achieve this, we observe that if $d_j \geqslant d_{j-1} + p_j$, then constraint $(j-1)$ of (4) implies constraint j of (4), i.e., $\sum_{i=1}^{j-1} p_i(1 - u_i) \leqslant d_{j-1}$ implies that $\sum_{i=1}^{j} p_i(1 - u_i) \leqslant d_j$. Thus, setting $d_j = \min\{d_j, d_{j-1} + p_j\}$ for $j = 2, \ldots, n$ leaves problems (P) and (LP) unaltered. With these reset due dates, immediately after executing Step 2 of the algorithm we have $\sum_{j \in S} p_j u_j^{(k)} \geqslant d^{(k)}$ since $\sum_{j \in S} p_j u_j^{(k)} \geqslant p_k \geqslant d_k - d_{k-1}$. Consequently, this iteration terminates with

$$\sum_{j=1}^{n} p_j u_j^{(k+1)} - \sum_{j=1}^{n} p_j u_j^{(k)} = d^{(k)}. \tag{11}$$

We next show that any optimal solution for problem (LP$^{(k-1)}$) is also an optimal solution for problem (LP$^{(k)}$) ($k = 2, \ldots, n$). To prove this, we first show that any feasible solution for problem (LP$^{(k-1)}$) is also a feasible solution for problem (LP$^{(k)}$). Since $u_j^{(k+1)} \geqslant u_j^{(k)}$ for $j = 1, \ldots, n$, it is clear that constraints (10) of (LP$^{(k-1)}$) imply constraints (10) of (LP$^{(k)}$). Also, constraints (8) of (LP$^{(k-1)}$) are identical with the first $k-2$ constraints of (8) for (LP$^{(k)}$). Using the fact that $u_j^{(k)} = 1$ for $j = 1, \ldots, k-1$, constraint (9) of (LP$^{(k-1)}$) can be written as

$$\sum_{j=1}^{k-1} p_j(1 - u_j) \leqslant d_{k-1} - \sum_{j=k}^{n} p_j(u_j^{(k)} - u_j). \tag{12}$$

Constraints (10) of (LP$^{(k-1)}$) are $u_j \leqslant u_j^{(k)}$ ($j = 1, \ldots, n$), which, when substituted into (12), yield $\sum_{j=1}^{k-1} p_j(1 - u_j) \leqslant d_{k-1}$, which is constraint $k-1$ of (8) for (LP$^{(k)}$). Therefore, each of the constraints (8) for (LP$^{(k)}$) is implied from the constraints of (LP$^{(k-1)}$). It remains to show that constraint (9) of (LP$^{(k)}$) may be deduced from the constraints of (LP$^{(k-1)}$). By writing constraint (9) of (LP$^{(k-1)}$) as

$$\sum_{j=1}^{n} p_j u_j \geqslant \sum_{j=1}^{n} p_j u_j^{(k)} - d_{k-1}, \tag{13}$$

and substituting (11) into (13) we obtain

$$\sum_{j=1}^{n} p_j u_j \geqslant \sum_{j=1}^{n} p_j u_j^{(k+1)} - d_k, \tag{14}$$

which is constraint (9) of ($LP^{(k)}$). We have now proved that any feasible solution of ($LP^{(k-1)}$) is also a feasible solution of ($LP^{(k)}$).

We show next that any feasible solution of ($LP^{(k)}$) also satisfies constraints (8) and (9) of ($LP^{(k-1)}$). Clearly constraints (8) of ($LP^{(k-1)}$) are satisfied since they are identical with the first $k-2$ constraints of (8) for ($LP^{(k)}$). Also, if (9) is satisfied for ($LP^{(k)}$), then it can be written as (14). Applying (11), we obtain (13) which is constraint (9) of ($LP^{(k-1)}$).

We now show by contradiction that any optimal solution of ($LP^{(k-1)}$) is also an optimal solution of ($LP^{(k)}$). Suppose that $u = u' = (u'_1, \ldots, u'_n)$ is an optimal solution of ($LP^{(k)}$) that is not optimal for ($LP^{(k-1)}$). If u' were feasible for ($LP^{(k-1)}$) then, since all feasible solutions for ($LP^{(k-1)}$) are feasible for ($LP^{(k)}$), it would be optimal for ($LP^{(k-1)}$). Therefore, $u = u'$ is infeasible for ($LP^{(k-1)}$). However, we have shown that if (8) and (9) of ($LP^{(k)}$) are satisfied, then (8) and (9) of ($LP^{(k-1)}$) are satisfied. Hence, u' must violate at least one of the constraints (10) of ($LP^{(k-1)}$), i.e., $u'_i > u_i^{(k)}$ for some i ($i = k, \ldots, n$). Consider first the case that $\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u'_j) < d_k - d_{k-1}$. Consider the solution $u'' = (u''_1, \ldots, u''_n)$ defined by $u''_j = u'_j$ ($j = 1, \ldots, n; j \neq i$) and $u''_i = u'_i - \epsilon$, where $\epsilon = \min\{u'_i - u_i^{(k)}, (d_k - d_{k-1} - \sum_{j=k}^{n} p_j(u_j^{(k+1)} - u'_j))/p_i\} > 0$. This choice of ϵ guarantees that $u''_i \geq u_i^{(k)} \geq 0$ and that

$$\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u''_j) \leq d_k - d_{k-1}. \tag{15}$$

Thus, since u' satisfies constraints (8) and (10) of ($LP^{(k)}$), u'' also satisfies these constraints. In particular, the last constraint of (8) is

$$\sum_{j=1}^{k-1} p_j(1 - u''_j) \leq d_{k-1}. \tag{16}$$

Adding (15) and (16) and using the fact that $u_j^{(k+1)} = 1$ for $j = 1, \ldots, k-1$ yields $\sum_{j=1}^{n} p_j(u_j^{(k+1)} - u''_j) \leq d_k$, verifying that u'' satisfies constraint (9) of ($LP^{(k)}$) and, hence, all constraints of ($LP^{(k)}$). Since $\sum_{j=1}^{n} w_j u''_j < \sum_{j=1}^{n} w_j u'_j$, u' cannot be an optimal solution which is a contradiction. Now consider the alternative case in which

$$\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u'_j) \geq d_k - d_{k-1}. \tag{17}$$

Consider the stage in the algorithm at which the final value of $u_i^{(k)}$ has just been set (either in Step 4 or Step 5). Suppose at this stage, for all jobs h with $h \in S - \{i\}$ we have $u'_h \geq u_h^{(k)}$. Noting that all jobs h ($h = k, \ldots, n$) with $h \notin S - \{i\}$ satisfy $u_h^{(k)} = 0$, we have $u'_h \geq u_h^{(k)}$ for $h = k, \ldots, i-1, i+1, \ldots, n$. Thus, since $u'_i > u_i^{(k)}$, we have

$$\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u'_j) < \sum_{j=k}^{n} p_j(u_j^{(k+1)} - u_j^{(k)}). \tag{18}$$

However, at each stage of the algorithm, we have

$$\sum_{j=k}^{n} p_j(u_j^{(k+1)} - u_j^{(k)}) \leq d^{(k)} = d_k - d_{k-1}. \tag{19}$$

Together (18) and (19) contradict (17). Thus, we assume that at the stage of the algorithm immediately after the final value of $u_i^{(k)}$ has been set, there exists some job $h \in S - \{i\}$ with $u'_h < u_h^{(k)}$. Also, from the method of selection from S by the algorithm and by our assumption of distinct ratios, we have $w_i/p_i > w_h/p_h$. Consider the solution $u'' = (u''_1, \ldots, u''_n)$ defined by $u''_j = u'_j$ ($j = 1, \ldots, n; j \neq h; j \neq i$), $u''_h = u'_h + \epsilon p_i/p_h$ and $u''_i = u'_i - \epsilon$, where $\epsilon = \min\{u'_i - u_i^{(k)}, (u_h^{(k)} - u'_h)p_h/p_i\} > 0$. This choice of ϵ ensures that $u''_h \leq u_h^{(k)} \leq u_h^{(k+1)}$ and that $u''_i \geq u_i^{(k)} \geq 0$. Thus, since u' satisfies constraints (8) and (10) of ($LP^{(k)}$), u'' also satisfies these constraints. To show that u'' also satisfies constraint (9) of ($LP^{(k)}$), we first observe that $\sum_{j=1}^{n} p_j u''_j = \sum_{j=1}^{n} p_j u'_j + p_h(u''_h - u'_h) + p_i(u''_i - u'_i) = \sum_{j=1}^{n} p_j u'_j$. Since u'' satisfies constraint (9) of ($LP^{(k)}$), it follows that u'' also satisfies this constraint. Therefore, u'' is a feasible solution of ($LP^{(k)}$). Now,

$$\sum_{j=1}^{n} w_j u''_j = \sum_{j=1}^{n} w_j u'_j + w_h(u''_h - u'_h) + w_i(u''_i - u'_i)$$

$$= \sum_{j=1}^{n} w_j u'_j + \epsilon(p_i w_h/p_h - w_i).$$

Since $w_i/p_i > w_h/p_h$, we deduce that $\sum_{j=1}^{n} w_j u''_j < \sum_{j=1}^{n} w_j u'_j$ which contradicts the assumption that u' is an optimal solution.

We have shown that any optimal solution for problem ($LP^{(k-1)}$) is an optimal solution for problem ($LP^{(k)}$) ($k = 2, \ldots, n$). This shows that any optimal solution for problem ($LP^{(1)}$) is an optimal solution for problem (LP). For problem ($LP^{(1)}$), constraint (9) is

$$\sum_{j=1}^{n} p_j(u_j^{(2)} - u_j) \leq d_1 = 0. \tag{20}$$

Constraints (10) of $(LP^{(1)})$ are $0 \leq u_j \leq u_j^{(2)}$ $(j = 1, \ldots, n)$. Consequently, from (20), the only feasible solution of $(LP^{(1)})$ is $u_j = u_j^{(2)}$ $(j = 1, \ldots, n)$ and hence it is the optimal solution. Clearly, the algorithm generates this optimal solution of $(LP^{(1)})$ which is also an optimal solution of (LP). QED

References

DANTZIG, G. B., "Discrete Variable Extremum Problems," *Oper. Res.*, 5 (1957), 266–277.

FAALAND, B., "A Weighted Selection Algorithm For Certain Tree-Structured Linear Programs," *Oper. Res.*, 32 (1984), 405–422.

INGARGIOLA, G. P. AND J. F. KORSH, "A Reduction Algorithm for Zero-One Single Knapsack Problems," *Management Sci.*, 20 (1973), 460–463.

KARP, R. M., "Reducibility among Combinatorial Problems," in *Complexity of Computations*, R. E. Miller and J. W. Thatcher (Eds.), Plenum Press, New York, 1972, 85–103.

LAWLER, E. L., "Sequencing to Minimize the Weighted Number of Tardy Jobs," *Rev. Francaise Automat. Recherche Operationnelle*, 10, 5 (1976), 27–33.

——— AND J. M. MOORE, "A Functional Equation and its Applications to Resource Allocation and Sequencing Problems," *Management Sci.*, 16 (1969), 77–84.

MARTELLO, S. AND P. TOTH, "The 0-1 Knapsack Problem," in *Combinatorial Optimization*, N. Christofides et al. (Eds.), Wiley, Chichester, 1979, 237–279.

MOORE, J. M., "An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," *Management Sci.*, 15 (1968), 102–109.

SAHNI, S. K., "Algorithms for Scheduling Independent Jobs," *J. Assoc. Comput. Mach.*, 23 (1976), 116–127.

VILLARREAL, F. J. AND R. L. BULFIN, "Scheduling a Single Machine to Minimize the Weighted Number of Tardy Jobs," *IE Trans.*, 15 (1983), 337–343.