"SINGLE MACHINE SCHEDULING TO
MINIMIZE TOTAL WEIGHTED
LATE WORK"

by

A.M.A. HARIRI,*
C.N. POTTS**
and
Luk VAN WASSENHOVE***

N° 92/29/TM

* Department of Statistics, King Abdul-Aziz University, Jeddah,
Saudi Arabia.

** Faculty of Mathematical Studies, University of Southampton, Southampton, England.

*** Professor of Operations Management and Operations Research, INSEAD, Boulevard de Constance,
Fontainebleau 77305 Cedex, France.

Printed at INSEAD,
Fontainebleau, France

SINGLE MACHINE SCHEDULING TO MINIMIZE TOTAL WEIGHTED LATE WORK

A.M.A. Hariri

Department of Statistics, King Abdul-Aziz University, Jeddah, Saudi Arabia

C.N. Potts

Faculty of Mathematical Studies, University of Southampton, U.K.

L.N. Van Wassenhove

INSEAD, Fontainebleau, France

In the problem of scheduling a single machine to minimize total weighted late work, there are n jobs to be processed for which each has an integer processing time, a weight and a due date. The objective is to minimize the total weighted late work, where the late work for a job is the amount of processing of this job which is performed after its due date. An $O(n \log n)$ algorithm is derived for the preemptive total weighted late work problem. For non-preemptive scheduling, efficient algorithms are derived for the special cases in which all processing times are equal and in which all due dates are equal. A pseudopolynomial algorithm is presented for the general non-preemptive total weighted late work problem. Also, a branch and bound algorithm in which lower bounds are obtained using the dynamic programming state-space relaxation method is proposed for this general problem. Computational results with the branch and bound algorithm for problems with up to 700 jobs are given.

Subject classification: Scheduling: single machine, weighted late work, branch and bound algorithm.

The problem of scheduling a single machine to minimize total weighted late work may be stated as follows. Each of n jobs (numbered $1, \ldots, n$) is to be processed on a single machine which can handle only one job at a time. Job i ($i = 1, \ldots, n$) becomes available for processing at time zero, requires a positive integer *processing time* p_i, has a positive *weight* w_i and has a positive integer *due date* d_i. In the preemptive version of this problem, processing may be interrupted and resumed at a later time, but in the non-preemptive problem no interruption in the processing of a job is allowed. We assume that jobs are numbered in non-decreasing order of their due dates (EDD order) so that $d_1 \leq \ldots \leq d_n$, and $w_i \geq w_{i+1}$ if $d_i = d_{i+1}$ ($i = 1, \ldots, n-1$). Given a schedule σ that defines the *completion time* $C_i(\sigma)$ of job i ($i = 1, \ldots, n$), the *late work* $V_i(\sigma)$ for job i, which is the amount of processing performed on i after its due date, is easily computed. When no ambiguity results, we abbreviate $C_i(\sigma)$ and $V_i(\sigma)$ to C_i and V_i respectively. If $V_i = 0$, then job i is *early*; if $0 < V_i < p_i$, then job i is *partially early*; alternatively, if $V_i = p_i$, then job i is *late*. We refer to $p_i - V_i$ as the *early work* for job i. The objective is to schedule the jobs so that the total weighted late work $\sum_{i=1}^{n} w_i V_i$ is minimized.

Problems which involve the scheduling of jobs with due dates on a single machine to minimize total cost have been widely studied in the literature. Two of these are related to the non-preemptive total weighted late work problem. Firstly, in the *total weighted tardiness problem*, which is (unary) NP-hard (Lawler [7], Lenstra et al. [9]), the cost of scheduling job i ($i = 1, \ldots, n$) to be completed at time C_i is given by its weighted tardiness $w_i T_i = w_i \max\{C_i - d_i, 0\}$. Clearly, $w_i V_i = w_i \min\{T_i, p_i\}$ for non-preemptive scheduling. An effective branch and bound algorithm which solves total weighted tardiness problems with up to 40 jobs is given by Potts and Van Wassenhove [10]. Secondly, in the *weighted number of late jobs problem*, the cost associated with job i is $w_i U_i$, where $U_i = 1$ if $C_i > d_i$ and $U_i = 0$ otherwise. A pseudopolynomial dynamic programming algorithm for this (binary) NP-hard problem (Karp [5]) is given by Lawler and Moore [8]. Potts and Van Wassenhove [11] present dynamic programming and branch and bound algorithms which solve weighted number of late jobs problems with up to 1000 jobs.

The total weighted late work problem is first studied by Blazewicz [2]. He shows that the problem of preemptively scheduling jobs with release dates on identical parallel machines can be solved using linear programming and is hence polynomially solvable. Potts and Van Wassenhove [12] derive an $O(n \log n)$ algorithm for the preemptive single machine problem with unit weights. They also show that the corresponding non-preemptive problem is (binary) NP-hard and they derive a pseudopolynomial dynamic programming algorithm which enables problems with up to 10 000 jobs to be solved.

The total weighted late work problem has applications in information processing. In this context, a job is a message carrying an amount of information proportional to its length. All information received after a given due date is useless and is referred to as information loss. Total weighted information loss is minimized by solving a total weighted late work problem. More generally, applications occur in any situation that involves a perishable commodity which deteriorates after a given due date.

This paper derives algorithms for the preemptive and non-preemptive total weighted late work problems. The remaining sections are organized as follows. Section 1 derives an $O(n \log n)$ algorithm for the preemptive problem, while subsequent sections deal with non-preemptive scheduling. Some special cases for which polynomial time algorithms are available are analyzed in Section 2. In Section 3, after deriving a key result on the structure of an optimal sequence, a pseudopolynomial dynamic programming algorithm is given. Subsequent sections of the paper are devoted to the derivation and computational testing of a branch and bound algorithm. Section 4 uses the recursion of Section 3 to derive, through the application of the dynamic programming state-space relaxation technique, a lower bounding scheme. Also included is the description of a heuristic which uses dynamic programming to generate an upper bound. Reduction tests, whereby jobs are discarded from the problem because they are either necessarily early or necessarily late, are given in Section 5. Section 6 analyzes various structural properties of the problem which enable the lower bounding scheme to be implemented more efficiently. Full details of our branch and bound algorithm are given in Section 7. Section 8 reports on computational experience with the algorithm and some concluding remarks are given in Section 9.

1. The preemptive problem

This section derives an $O(n \log n)$ algorithm for the preemptive scheduling problem. For the preemptive problem, whenever a job is being processed at its due date, it is preempted at this point and processing resumes at some arbitrary later time after the largest due date. Thus, to specify a solution of the problem, it is sufficient to find a schedule of early work up to the largest due date, since any remaining processing can then be arbitrarily scheduled as late work.

Our algorithm uses backward scheduling, where at each decision point t, early work on a job having the largest weight is scheduled. In the formal description below, S represents the set of jobs for which some processing remains to be scheduled and the current value of p_i represents the unscheduled processing for job i ($i = 1, \ldots, n$).

Preemptive Scheduling Algorithm

Step 1. Renumber the jobs in EDD order, set $S = \{1, \ldots, n\}$ and set $t = d_n$.

Step 2. Find the set $A_t = \{i | i \in S; d_i \geq t\}$ of jobs available to be completed at time t and choose $j \in A_t$ with w_j as large as possible. If possible, choose job k, with k chosen as large as possible, such that $t - p_j < d_k < t$ and set $s = d_k$; otherwise set $s = \max\{t - p_j, 0\}$.

Step 3. Schedule $t - s$ units of processing of job j in the interval $[s, t]$, set $p_j = p_j - (t - s)$ and set $t = s$. If $p_j = 0$, set $S = S - \{j\}$. If $S = \emptyset$ or $t = 0$, compute the total weighted late work $\sum_{i=1}^{n} w_i p_i$ and stop. Otherwise, if $d_i < t$ for all $i \in S$, set $t = \max_{i \in S}\{d_i\}$. Go to Step 2.

We claim that the Preemptive Scheduling Algorithm generates at most n preemptions. To justify our claim, we note that a preemption occurs when less than p_j units of processing of job j are scheduled in Step 3. This occurs either when $t - p_j < d_k < t$ for some k ($k = 1, \ldots, n-1$) or when $t - p_j < 0 < t$; thus, our claim is established. If desired, the number of preemptions may be reduced to at most $n-1$ by rescheduling all early work in EDD order and eliminating any preemption at time d_n by appropriate scheduling of late work to start at time d_n.

Since at most n preemptions are generated by the Preemptive Scheduling Algorithm, Steps 2 and 3 of the algorithm are executed at most $2n$ times. It is apparent, therefore, that the algorithm requires $O(n \log n)$ time. Note also that the Preemptive Scheduling Algorithm remains valid if processing times and due dates become arbitrary positive real numbers.

Lastly in this section, we prove that the algorithm generates an optimal solution.

Theorem 1. *The schedule generated by the Preemptive Scheduling Algorithm minimizes the total weighted late work.*

Proof. Consider an optimal schedule π^* and compare it with a schedule π^A generated by the Preemptive Scheduling Algorithm. We assume that in π^*, any processing scheduled before time d_n is early work; otherwise it can be rescheduled after time d_n without affecting the total late work. It is also assumed that π^* has a finite number of preemptions: we observe that an optimal schedule obtained from the linear programming algorithm of Blazewicz has this property. We show that a finite sequence of transformations of π^*, each of which does not increase the total weighted late work, yields the schedule π^A.

Suppose that π^* and π^A are not identical in the time interval $[0, d_n]$. Let t be chosen as small as possible so that these schedules are identical in the interval $[t, d_n]$. In π^A, suppose that processing on job j is scheduled in the interval $[s, t]$ and not processed immediately before time s. (The machine cannot be idle just before time t in the schedule π^A because if it were, the mechanics of the Preemptive Scheduling Algorithm ensure that no early work is available, so the machine would also be idle in π^* immediately before time t, thereby contradicting the definition of t.) Consider first the case that in π^*, the machine is idle in the interval $[r, t]$, but is not idle immediately before time r. Let $q = \max\{r, s\}$. We transform π^* by moving $t - q$ units of processing of job j into the interval $[q, t]$, where this processing is originally scheduled before time q or after time d_n. Since the processing moved into $[q, t]$ is completed by time d_j (the Preemptive Scheduling Algorithm would not schedule job j in the interval $[s, t]$ otherwise), this transformation does not increase the total weighted late work. We now consider the alternative case that in π^*, job i ($i \neq j$) is processed in the interval $[r, t]$ and not processed immediately before time r. We transform π^* by interchanging the $t - q$ units of processing of job i scheduled in $[q, t]$ with $t - q$ units of processing of job j which are originally scheduled before time q or after time d_n. Since, in the Preemptive Scheduling Algorithm, jobs i and

j are candidates for scheduling immediately before time t, job j is selected because $w_j \geq w_i$. Any interchange of processing of job i with early work for job j (scheduled before time q) leaves the total weighted late work unaltered. Furthermore, any interchange of processing of job i with processing which corresponds to late work for job j decreases the total weighted late work by $w_j - w_i$ per unit of interchange. Thus, this transformation does not increase the total weighted late work either. In both cases, the transformed schedule, which is also optimal, is identical with π^A in the interval $[q, d_n]$. Since π^* and π^A each have a finite number of preemptions, repetition of this argument shows that an optimal schedule π^A is obtained after a finite number of transformations of π^*. □

2. Special cases

Henceforth, we consider non-preemptive scheduling. In this section, we present an $O(n)$ algorithm for the case of identical due dates and an $O(n^3)$ algorithm for the case of identical processing times.

We consider first the case of *identical due dates* for which $d_i = d$ for $i = 1, \ldots, n$, where d is a positive integer. We assume that $d < \sum_{i=1}^{n} p_i$; otherwise, any sequence is optimal. Recall that jobs with equal due dates are numbered in non-increasing order of weights. Thus, $w_1 \geq \ldots \geq w_n$. The following result provides a class of optimal sequences.

Theorem 2. *For the case of identical due dates, if j is chosen so that $\sum_{i=1}^{j-1} p_i < d \leq \sum_{i=1}^{j} p_i$, then any sequence in which jobs $1, \ldots, j-1$ are scheduled before job j and jobs $j+1, \ldots, n$ are scheduled after job j is optimal.*

Proof. Suppose that the Preemptive Scheduling Algorithm is applied. It schedules all the processing of jobs $1, \ldots, j-1$ and $d - \sum_{i=1}^{j-1} p_i$ units of processing of job j as early work. By scheduling jobs $1, \ldots, j-1$ (in any order) in the interval $[0, \sum_{i=1}^{j-1} p_i]$ and then the early work for job j in the interval $[\sum_{i=1}^{j-1} p_i, d]$, we still have an optimal preemptive schedule. A non-preemptive schedule is obtained by appending any late work for job j, followed by jobs $j+1, \ldots, n$ (in any order). Since this non-preemptive schedule has the same total weighted late work as the optimal preemptive schedule, it is an optimal non-preemptive schedule. □

To complete our analysis of the case of identical due dates, we show that job j of Theorem 2 can be found (without renumbering the jobs) in $O(n)$ time. We assume that all weights are distinct: this can be achieved, if necessary, by perturbing the weights. Let w_i be the median weight which is found in $O(n)$ time (Blum et al. [3], Schonhage et al. [13]). Also, let $A^- = \{h | w_h < w_i\}$, $A = \{i\}$ and $A^+ = \{h | w_h > w_i\}$. Either $\sum_{h \in A^+} p_h \geq d$ in which case the search for job j is restricted to A^+ while the jobs of A^- and A are late; or $\sum_{h \in A^-} p_h > \sum_{h=1}^{n} p_h - d$ in which case the search for job j is restricted to A^- while the jobs of A and A^+ are early; or $\sum_{h \in A^+} p_h < d \leq \sum_{h=1}^{n} p_h - \sum_{h \in A^-} p_h$ in which case the search ends with

$j = i$ while the jobs of A^+ are early and the jobs of A^- are late. To continue the search in the former cases, late jobs are discarded immediately, whereas early jobs are discarded after their processing times are subtracted from d. Furthermore, the search for job j is restricted to a subset containing at most $n/2$ jobs. Reapplying the procedure requires one half of the time required by the first application and restricts the search to a subset containing at most $n/4$ jobs. After $O(\log n)$ applications of the procedure which are carried out over subsets which contain at most n, $n/2$, $n/4$, ... jobs, job j is found in $O(n)$ time. It is now clear from Theorem 2 that the case of identical due dates is solvable in $O(n)$ time.

We consider now the case of *identical processing times* for which $p_i = p$ for $i = 1, \ldots, n$, where p is a positive integer. If job i ($i = 1, \ldots, n$) is sequenced in position j ($j = 1, \ldots, n$), then it has completion time jp from which its weighted late work is given by

$$c_{ij} = w_i \min\{\max\{jp - d_i, 0\}, p_i\}.$$

It is now apparent that, using the algorithm of Lawler [6], an optimal sequence may be found in $O(n^3)$ time from the solution of a linear assignment problem with costs c_{ij}. Thus, the case of equal processing times is solvable in $O(n^3)$ time.

3. A dynamic programming algorithm

Henceforth, the general problem of non-preemptively scheduling jobs on a single machine to minimize the total weighted late work is considered. In this section, we derive a pseudopolynomial dynamic programming algorithm for this problem. The algorithm relies on the closeness of an optimal schedule of early and partially early jobs to an EDD sequence; this issue is discussed first. Consider the following instance. There are two jobs for which $p_1 = 3$, $w_1 = 1$, $d_1 = 5$, $p_2 = 4$, $w_2 = 3$ and $d_2 = 6$. The total weighted late work for the sequence $(2, 1)$ is 2, which is less than the value of 3 that is given by EDD sequence $(1, 2)$. Thus, in contrast to the total late work problem where $w_i = 1$ ($i = 1, \ldots, n$), we cannot assume that early and partially early jobs are sequenced in EDD order.

Clearly, an optimal schedule is specified by a sequence of early and partially early jobs which we call a *non-late sequence*; any late jobs can be appended to this sequence in an arbitrary order. Job i is *deferred* (from its EDD position) in a non-late sequence σ if it is sequenced after a job having a due date larger than d_i. If job i is sequenced immediately after job j in σ, where $d_j > d_i$, then ji forms a *reversed pair* in σ.

The following result establishes the ordering of jobs with the same due date in an optimal non-late sequence σ. It also shows we may assume that jobs of σ are almost in EDD order in the sense that for each job j of σ, at most one job with a smaller due date is sequenced after it.

Theorem 3. *There exists an optimal non-late sequence σ such that jobs of σ with the same due date are sequenced in non-increasing order of their weights and, for*

each job j of σ, at most one job having a due date smaller than d_j appears after j in σ.

Proof. Let σ be any optimal non-late sequence. We show that a finite sequence of transformations of σ, each of which does not increase the total weighted late work, yields an optimal non-late sequence which satisfies the conditions of the theorem. Suppose that σ does not satisfy the conditions of the theorem. We consider first the case that some job j is sequenced before job i in σ, where $d_i = d_j$ and $w_i > w_j$. It is apparent that job j is early in σ since if it were partially early, then job i would be late. Firstly, suppose that $V_i(\sigma) \geq p_j$. Consider the effect of removing job j from σ so that it is deemed to be late. The weighted late work for job i decreases by $w_i p_j$ to give a decrease in the total weighted late work of at least $p_j(w_i - w_j) > 0$. However, this contradicts the choice of σ as an optimal non-late sequence. Therefore, $0 \leq V_i(\sigma) < p_j$. Consider now the non-late sequence σ' which is obtained from σ by removing job j from its original position and inserting it immediately after job i. Clearly, $V_i(\sigma') = 0$ and $V_j(\sigma') = V_i(\sigma)$. Furthermore, only job j has a later completion time in σ' than in σ. Thus, we deduce that

$$w_i V_i(\sigma) + w_j V_j(\sigma) - w_i V_i(\sigma') - w_j V_j(\sigma') = (w_i - w_j) V_i(\sigma) \geq 0$$

which shows that σ' is an alternative optimal non-late sequence. By repeating this argument, an optimal non-late sequence results in which jobs having the same due date are sequenced in non-increasing order of their weights.

Alternatively, suppose that σ is an optimal non-late sequence in which jobs having the same due date are sequenced in non-increasing order of their weights, but does not satisfy the conditions of the theorem. Choose the last job j of σ which is sequenced before jobs i and i', where $d_i < d_j$ and $d_{i'} < d_j$. We assume without loss of generality that job i is sequenced before job i' in σ. From the inequality $C_i(\sigma) < d_{i'}$, which is valid since otherwise job i' would be late in σ, we deduce that $C_i(\sigma) < d_j$. Consider now the non-late sequence σ' which is obtained from σ by removing job j from its original position and inserting it immediately after job i. We observe that $C_j(\sigma') = C_i(\sigma) < d_j$. Since no job is completed later in σ' than in σ except job j which has zero late work in σ and σ', it is clear that σ' is also an optimal non-late sequence. Furthermore, σ' also has the property that jobs with the same due date are sequenced in non-increasing order of their weights. By repeating this argument, an optimal sequence satisfying the conditions of the theorem is obtained after a finite sequence of transformations of σ. □

We now proceed with the derivation of our pseudopolynomial dynamic programming algorithm. It generalizes the algorithms of Lawler and Moore for the problem of minimizing the weighted number of late jobs and of Potts and Van Wassenhove [12] for the total late work problem.

We first show how Theorem 3 allows us to perform a structured search for an optimal non-late sequence. We restrict our search to *EDD-maximal* non-late sequences for which the interchange of jobs j and i in any reversed pair ji results in

an increase in the total weighted late work. Note that if ji is a reversed pair in an EDD-maximal non-late sequence σ, then job j is early and job i is partially early. To justify this assertion, we observe that if job j is completed after time d_j, then $d_j > d_i$ shows job i to be late and hence not in σ; thus, job j is early. Furthermore, if job i is early, then jobs j and i remain early if they are interchanged which implies that σ is not EDD-maximal; thus, job i is partially early. Thus, if job i of the reversed pair ji is completed at time t, then $i \in A_{jt}$, where

$$A_{jt} = \{i | d_i < d_j;\ d_i < t < d_i + p_i;\ i = 1, \ldots, j-1\}.$$

In our structured search, jobs are considered in natural (EDD) order $1, \ldots, n$. Job i can either be declared late, be declared early or partially early and not deferred, or be deferred. Suppose that job i is deferred to be sequenced immediately after job j, where $d_j > d_i$ and consequently $j > i$. Theorem 3 ensures that all non-late jobs amongst $i+1, \ldots, j$ are sequenced in their natural order. Thus, at any stage of the procedure, there is at most one job which is deferred, but not currently sequenced.

Our dynamic programming algorithm utilizes this search procedure. In addition to the variable j which indicates that jobs $1, \ldots, j$ only are considered, the algorithm uses a state-space that consists of (t, i). The last element specifies which job, if any, is deferred: if $i = 0$, there is no deferred job; however $i \in \{1, \ldots, j\}$ indicates that job i is deferred and will be sequenced immediately after one of the jobs $j+1, \ldots, n$. Also, t represents the time at which non-late jobs amongst $\{1, \ldots, j\} - \{i\}$ are completed. The dynamic programming recursion is defined on values $f_j(t, i)$ which represent the minimum total weighted late work for jobs $1, \ldots, j$ when non-late jobs amongst $\{1, \ldots, j\} - \{i\}$ are completed at time t, where any deferred job i contributes $w_i p_i$ in the computation of $f_j(t, i)$. (An equivalent algorithm can be derived in which a deferred job i has a zero contribution in $f_j(t, i)$.) Thus, for each j $(j = 1, \ldots, n)$, $f_j(t, i)$ is defined for $t = 0, \ldots, T_j$ and $i = 0, \ldots, j$, where $T_j = \sum_{h=1}^{j} p_h$.

Our recursion equations are:

$$f_j(t,0) = \begin{cases} \min\{f_{j-1}(t,0) + w_j p_j,\ f_{j-1}(t - p_j, 0) \\ \quad + w_j \max\{t - d_j, 0\},\ \min_{i \in A_{jt}}\{f_{j-1}(t - p_j - p_i, i) \\ \qquad\qquad + w_i(t - d_i - p_i)\}\} & \text{for } t < d_j + p_j, \\ \min\{f_{j-1}(t,0) + w_j p_j,\ \min_{i \in A_{jt}}\{f_{j-1}(t - p_j - p_i, i) \\ \qquad\qquad + w_i(t - d_i - p_i)\}\} & \text{for } t \geq d_j + p_j; \end{cases}$$

$$f_j(t,i) = \begin{cases} \min\{f_{j-1}(t,i) + w_j p_j,\ f_{j-1}(t - p_j, i)\} & \text{for } t < d_i, \\ \infty & \text{for } t \geq d_i; \end{cases} \quad (i = 1, \ldots, j-1)$$

$$f_j(t,j) = \begin{cases} f_{j-1}(t,0) + w_j p_j & \text{for } t < d_j, \\ \infty & \text{for } t \geq d_j; \end{cases}$$

where $f_0(0,0) = 0$ and all other initial values are set to infinity.

Some explanation of these recursion equations is appropriate. In the computation of $f_j(t,0)$ for $t < d_j + p_j$, the three terms in the minimization correspond to the decisions that job j is late, job j is scheduled in the interval $[t-p_j, t]$ to be non-late, and job j and the deferred job i are sequenced successively in the interval $[t-p_j-p_i, t]$. In the latter case, the definition of A_{jt} ensures that job j is early and job i is partially early, while the weighted late work $w_i(t-d_i)$ of job i is added and the contribution $w_i p_i$ assumed in $f_{j-1}(t-p_j-p_i, i)$ is subtracted from the function value. When $t \geq d_j + p_j$, the computation is similar except that the second term, which assumes that job j is non-late when completed at time t, is deleted. For $f_j(t,i)$, which is defined for $t < d_i$ since otherwise job i must be late, the cases that job j is late or is early are considered (job j cannot be partially early when completed at time t since $t < d_i \leq d_j$). Finally, the computation of $f_j(t,j)$, which is defined for $t < d_j$ since otherwise job j must be late, sets job j to be deferred.

The recursion equations for $f_j(t,i)$ are solved for $j = 1, \ldots, n$, $t = 0, \ldots, T_j$ and $i = 0, \ldots, j$ after which $\min_{t=0,\ldots,T_n}\{f_n(t,0)\}$ provides the minimum total weighted late work. Since $|A_{jt}| < n$, our dynamic programming algorithm requires $O(n^2 T_n)$ time. This establishes that the total weighted late work problem is pseudopolynomially solvable because $T_n = \sum_{h=1}^{n} p_h$.

In Sections 5 and 6, various devices are presented which help to reduce the time and storage requirements of the dynamic programming algorithm. Even with these devices, however, the algorithm is awkward to implement and storage requirements are substantial. As an alternative, we propose a branch and bound algorithm in which the lower bounds are derived, using the dynamic programming state-space relaxation method, from the recursion given above. The remainder of the paper is devoted to a description and an evaluation of our branch and bound algorithm.

4. Lower and upper bounds

In this section, dynamic programming is used to establish lower and upper bounds on the minimum total weighted late work; these bounds are used in our branch and bound algorithm. Firstly, we derive a lower bounding scheme by applying the dynamic programming state-space relaxation method to the recursion of the previous section. Dynamic programming state-space relaxation is a technique proposed by Christofides et al. [4] for routing problems and is developed by Abdul-Razaq and Potts [1] for single machine scheduling. The method maps the state-space of a dynamic programming recursion onto a smaller state-space and computes a lower bound by performing the recursion on the smaller state-space.

In addition to the state variable j, the recursion of the previous section uses a state-space consisting of (t,i), where the value of i defines any deferred job. We relax this state-space by mapping (t,i) onto t, i.e., the mapping discards the state variable defining any deferred job.

Our relaxed dynamic programming recursion is defined on values $f'_j(t)$. To ensure that a valid lower bound is obtained, we require that $f'_j(t) \leq \min_{i=0,\ldots,j}\{f_j(t,i)\}$.

By defining

$$f'_j(t) = \begin{cases} \min\{f'_{j-1}(t) + w_j p_j, f'_{j-1}(t-p_j) + w_j \max\{t-d_j,0\}, \\ \qquad \min_{i\in A_{jt}}\{f'_{j-1}(t-p_j-p_i) + w_i(t-d_i-p_i)\}\} & \text{for } t < d_j + p_j, \\ \min\{f'_{j-1}(t) + w_j p_j, \min_{i\in A_{jt}}\{f'_{j-1}(t-p_j-p_i) \\ \qquad +w_i(t-d_i-p_i)\}\} & \text{for } t \geq d_j + p_j, \end{cases}$$

where $f'_0(0) = 0$ and all other initial values are set to infinity, a straightforward inductive argument shows that this requirement is satisfied. We note that in the computation of $f'_j(t)$, decisions may be taken to schedule a job twice, the first time in its EDD position and the second time as a deferred job.

The recursion equations for $f'_j(t)$ are solved for $j = 1,\ldots,n$ and $t = 0,\ldots,T_j$ after which the lower bound is computed using $\text{LB} = \min_{t=0,\ldots,T_n}\{f'_n(t)\}$. The computation of the lower bound requires $O(n^2T_n)$ time. A backtracking procedure yields the corresponding non-late 'sequence'. If this 'sequence' contains no repeated job, then it is optimal. Otherwise, the branch and bound algorithm described in Section 7 is applied.

Included in Section 6 is the derivation of a procedure that allows jobs to be eliminated from A_{jt}. Results of initial experiments show $|A_{jt}|$ to be very small for all j and t, after this procedure is applied. Therefore, for practical purposes, our recursion for computing lower bounds resembles an $O(nT_n)$ procedure, and is more efficient than the $O(n^2T_n)$ dynamic programming algorithm of Section 3.

The following *dynamic programming heuristic* is applied at the root node of the search tree in our branch and bound algorithm. Our heuristic computes an optimal solution to the problem in which deferred jobs are not allowed, i.e., early and partially early jobs are constrained to be sequenced in EDD order. Let $g_j(t)$ denote the minimum total weighted late work incurred when scheduling jobs $1,\ldots,j$ so that early and partially early jobs are sequenced in EDD order and the last of these non-late jobs is completed at time t. From the equations of Section 3 for $f_j(t,0)$, we obtain the following recursion:

$$g_j(t) = \begin{cases} \min\{g_{j-1}(t) + w_j p_j, g_{j-1}(t-p_j) \\ \qquad +w_j \max\{t-d_j,0\}\} & \text{for } t < d_j + p_j \\ g_{j-1}(t) + w_j p_j & \text{for } t \geq d_j + p_j \end{cases}$$

where $g_0(0) = 0$ and all other initial values are set to infinity. After computing $g_j(t)$ for $j = 1,\ldots,n$ and $t = 0,\ldots,T_j$, we obtain $\text{UB} = \min_{t=0,\ldots,T_n}\{g_n(t)\}$ as the weighted late work for our approximate solution. Clearly, this heuristic requires $O(nT_n)$ time. The substantial computational requirements are partly justified by the results of initial experiments which show that the the heuristic generates an optimal solution in many cases.

The two following sections present various devices which help to improve the effectiveness of our branch and bound algorithm. Section 5 describes reduction tests whereby jobs can be discarded from the problem. Also, we give a redundant

state elimination procedure which allows the recursions of this section to be solved more efficiently. In Section 6, we derive a reversed pair elimination procedure which restricts the cardinality of the sets A_{jt}: in addition to reducing the computation time for LB, there is a decreased likelihood that the 'sequence' corresponding to LB contains repeated jobs and consequently the lower bound becomes tighter.

5. Reduction tests

5.1 Earliness test

For our earliness test, we establish conditions whereby jobs can be removed from the problem because they are necessarily early when sequenced after all other early and partially early jobs. Specifically, we aim to show, for some index j, that jobs $j+1,\ldots,n$ are early in at least one optimal schedule; we choose j as small as possible, subject to the conditions of the test. For the problem involving jobs $1,\ldots,i$ only ($i=1,\ldots,n$), we define recursively the *latest completion time* T_i^L of the last early or partially early job using

$$T_i^L = \min\{T_{i-1}^L + p_i, \max_{h\in\{1,\ldots,i\}}\{d_h + p_h - 1\}\}, \tag{1}$$

where $T_0^L = 0$. (A justification for the use of this recursion is given below in the proof of Theorem 4.) If jobs $j+1,\ldots,n$ are necessarily early when sequenced in EDD order after the last early or partially early job amongst $1,\ldots,j$, then clearly $j+1,\ldots,n$ are early in at least one optimal schedule. The earliness test selects j as small as possible, subject to

$$T_j^L + \sum_{h=j+1}^{k} p_h \le d_k \text{ for } k = j+1,\ldots,n, \tag{2}$$

and discards jobs $j+1,\ldots,n$ from the problem. We show next that an optimal solution is obtained by appending jobs $j+1,\ldots,n$ to an optimal non-late sequence for the reduced problem.

Theorem 4. *An optimal solution is obtained by applying the earliness test, solving the reduced problem, and then inserting jobs $j+1,\ldots,n$ immediately after the last early or partially early job of the optimal sequence for the reduced problem.*

Proof. If the last early or partially early job of the reduced problem is completed by time T_j^L, then clearly (2) allows jobs $j+1,\ldots,n$ to be inserted without increasing the total weighted late work. Thus, it is sufficient to show that the last early or partially early job amongst $1,\ldots,j$ is completed not later than time T_j^L.

We consider three cases. Firstly, if $T_j^L = \sum_{i=1}^{j} p_i$, then it is clear that jobs $1,\ldots,j$ are completed by time T_j^L. Secondly, if $T_j^L = \max_{i=1,\ldots,j}\{d_i + p_i - 1\}$, then

any of the jobs $1,\ldots,j$ is late if it is completed after time T_j^L. Finally, suppose that $T_j^L = \max_{h=1,\ldots,i}\{d_h + p_h - 1\} + \sum_{h=i+1}^{j} p_h$ for some job i $(i = 1,\ldots,j-1)$. The last early or partially early job amongst $1,\ldots,i$ is completed not later than time $\max_{h=1,\ldots,i}\{d_h + p_h - 1\}$. Furthermore, after this last job is completed, a maximum of $\sum_{h=i+1}^{j} p_h$ additional units of processing are required to complete any further early or partially early jobs amongst $1,\ldots,j$. Thus, the desired result holds for all three cases. □

5.2 Lateness test

Our lateness test gives conditions under which jobs can be removed from the problem because they are necessarily late in any optimal schedule. Let LB_j be a lower bound on the total weighted late work when job j is constrained to be early or partially early. (The computation of LB_j using the Preemptive Scheduling Algorithm is explained below.) If $LB_j > UB$, where UB is obtained by applying our dynamic programming heuristic of Section 4 (although any other upper bound on the total weighted late work can be used), then in any optimal schedule job j must be late. Thus, after the evaluation of its contribution $w_j p_j$ to the total weighted late work, job j is discarded to give a reduced problem. The test is applied for each job j except those which, in the solution generated by the Preemptive Scheduling Algorithm applied to the original problem, are early.

The computation of the lower bound LB_j using the Preemptive Scheduling Algorithm is described now. Since job j is constrained to be early or partially early, it must be completed no later than time $d_j + p_j - 1$. To enforce this constraint, we assign a large weight to job j and reset its due date to $d_j + p_j - 1$. The value LB_j is the minimum total weighted late work for the preemptive problem having processing times $p'_i = p_i$ $(i = 1,\ldots,n)$, weights $w'_i = w_i$ $(i = 1,\ldots,n;\ i \neq j)$ and $w'_j = \infty$, and due dates $d'_i = d_i$ $(i = 1,\ldots,n;\ i \neq j)$ and $d'_j = d_j + p_j - 1$.

The following results justifies the use of our lateness test.

Theorem 5. *An optimal solution is obtained by applying the lateness test to eliminate some set L of jobs, solving the reduced problem, and then appending the jobs of L to the optimal sequence for the reduced problem.*

Proof. For a fixed job j, it is sufficient to establish that LB_j is a valid lower bound on the total weighted late work for the (non-preemptive) constrained problem. Let π denote an optimal sequence for this problem which defines $V_i(\pi)$ as the late work for job i $(i = 1,\ldots,n)$. Suppose that π is evaluated with respect to the data for the preemptive problem to give $V'_i(\pi)$ as the late work for each job i. Clearly, $V'_i(\pi) = V_i(\pi)$ for $i = 1,\ldots,n$ and $i \neq j$. Since by the constraint on job j we have $C_j(\pi) \leq d_j + p_j - 1$, it follows that $V'_j(\pi) = 0 \leq V_j(\pi)$. Therefore, $\sum_{i=1}^{n} w_i V_i(\pi) \geq \sum_{i=1}^{n} w_i V'_i(\pi) \geq LB_j$, where the final inequality holds because LB_j is the minimum total weighted late work for the preemptive problem. It is now apparent that LB_j is a valid lower bound for use in our lateness test. □

5.9 Redundant state elimination

In this subsection, we aim to improve the efficiency of the computation of the lower and upper bounds described in Section 4. In particular, we show that the range of values of t for which $f_j'(t)$ and $g_j(t)$ are computed can be restricted through *redundant state elimination.* For each state variable j, we use the definition of latest completion times in Section 5.1 to eliminate states for which $t > T_j^L$. Thus, it is sufficient to compute $f_j'(t)$ and $g_j(t)$ for $j = 1, \ldots, n$ and $t = 0, \ldots, T_j^L$.

Further state variables are eliminated as follows. Let T_i^E $(i = 1, \ldots, n-1)$ denote the latest start time for the processing of jobs $i+1, \ldots, n$ if each is to be completed by its due date (when sequenced in EDD order). It is clear that T_i^E is computed from the backward recursion

$$T_i^E = \min\{T_{i+1}^E, d_{i+1}\} - p_{i+1},$$

where $T_n^E \geq d_n$. It is convenient to set $T_n^E = T_n^L$, where T_n^L is obtained from (1). After our earliness test is applied, it is easily verified that $T_n^L \geq d_n$ and $T_i^E < T_i^L$ $(i = 1, \ldots, n-1)$.

When $T_j^E \geq 0$ for any job j, our redundant state elimination procedure restricts the range of values of t for which $f_j'(t)$ and $g_j(t)$ are required in the solution of subsequent recursion equations. More precisely, under the assumptions that define the function values $f_j'(t)$ and $g_j(t)$, the values of the minimum total weighted late work for any schedule in which jobs $j+1, \ldots, n$ are early are respectively

$$U_j = \min_{t=0,\ldots,T_j^E}\{f_j'(t)\}, \qquad V_j = \min_{t=0,\ldots,T_j^E}\{g_j(t)\}.$$

If schedules having total weighted late work less than U_j and V_j are to be found, jobs $j+1, \ldots, n$ cannot each be early. In such schedules, the last early or partially early job amongst $1, \ldots, j$ is, therefore, completed later than time T_j^E. Thus, after calculation of U_j and V_j, we assume $f_j'(t) = \infty$ and $g_j(t) = \infty$ for $t = 0, \ldots, T_j^E$, and store $f_j'(t)$ and $g_j(t)$ only for $t = T_j^E + 1, \ldots, T_j^L$. We note that the indices $t = 0, \ldots, T_j^E$ are not considered further since the relevant recursions show that $f_k'(t) = \infty$ and $g_k(t) = \infty$ for $t = 0, \ldots, T_j^E$ when $k > j$. After U_n and V_n are found, we compute the lower and upper bounds $\mathrm{LB} = \min_{j=1,\ldots,n}\{U_j | T_j^E \geq 0\}$ and $\mathrm{UB} = \min_{j=1,\ldots,n}\{V_j | T_j^E \geq 0\}$.

6. Reversed pair elimination

The analysis in this section establishes conditions under which ji cannot form a reversed pair in an optimal EDD-maximal non-late sequence. These conditions are used to eliminate jobs from the sets A_{jt}. In addition to reducing computational requirements for our lower bound, a reduction in $|A_{jt}|$ often leads to a tighter lower bound since the likelihood is reduced that the corresponding 'sequence' contains repeated jobs.

Lemma 1. *If σ is any non-late sequence containing a reversed pair ji, then σ is not EDD-maximal if at least one of the following conditions is satisfied:*
(a) $C_i(\sigma) \leq d_j$;
(b) $w_i \geq w_j$;
(c) $C_i(\sigma) \leq \min\{(w_j d_j - w_i d_i)/(w_j - w_i), d_i + p_j\}$;
(d) $d_i + p_j < C_i(\sigma) \leq d_j + w_i p_j / w_j$.

Proof. Let σ' be the non-late sequence obtained from σ by interchanging jobs i and j, and let $V(\sigma)$ and $V(\sigma')$ denote the total weighted late work associated with σ and σ' respectively. We establish, whenever at least one of (a), (b), (c) and (d) holds, that σ is not EDD-maximal by showing $V(\sigma) - V(\sigma') \geq w_i V_i(\sigma) - w_i V_i(\sigma') - w_j V_j(\sigma') \geq 0$ which implies $V(\sigma) \geq V(\sigma')$.

Consider first the case that condition (a) is satisfied. Clearly, $C_j(\sigma') = C_i(\sigma) \leq d_j$ yields $V_j(\sigma') = 0$. Also, since $C_i(\sigma') < C_i(\sigma)$ we have $V_i(\sigma') \leq V_i(\sigma)$. We deduce that $V(\sigma) - V(\sigma') \geq w_i(V_i(\sigma) - V_i(\sigma')) \geq 0$ when condition (a) is satisfied.

It remains to be shown that the lemma holds when $C_i(\sigma) > d_j$ and at least one of (b), (c) and (d) is satisfied. When $C_i(\sigma) > d_j$, the condition $d_j > d_i$ shows job i cannot be early in σ. Thus, it is partially early and

$$V_i(\sigma) = C_i(\sigma) - d_i. \tag{3}$$

Since $C_j(\sigma') = C_i(\sigma)$, by definition we have

$$V_j(\sigma') = \min\{C_i(\sigma) - d_j, p_j\} \leq C_i(\sigma) - d_j. \tag{4}$$

Noting that $C_i(\sigma') = C_i(\sigma) - p_j$, if $C_i(\sigma) - p_j \leq d_i$, we have

$$V_i(\sigma') = 0. \tag{5}$$

Alternatively, if $C_i(\sigma) - p_j \geq d_i + 1$, we have

$$V_i(\sigma') = C_i(\sigma) - p_j - d_i. \tag{6}$$

Firstly, suppose that $C_i(\sigma) \leq d_i + p_j$. From (3), (4) and (5) we obtain

$$V(\sigma) - V(\sigma') \geq w_i(C_i(\sigma) - d_i) - w_j(C_i(\sigma) - d_j)$$

which shows that $V(\sigma) - V(\sigma') \geq 0$ either when $w_i \geq w_j$ or when $w_i < w_j$ and $C_i(\sigma) \leq (w_j d_j - w_i d_i)/(w_j - w_i)$.

Alternatively, suppose that $C_i(\sigma) > d_i + p_j$. Using the inequality $V_j(\sigma') \leq p_j$ together with (3) and (6) we obtain

$$V(\sigma) - V(\sigma') \geq (w_i - w_j)p_j$$

which shows that $V(\sigma) - V(\sigma') \geq 0$ when $w_i \geq w_j$. Also, we deduce from (3), (4) and (6) that

$$V(\sigma) - V(\sigma') \geq w_i p_j - w_j(C_i(\sigma) - d_j)$$

which shows that $V(\sigma) - V(\sigma') \geq 0$ when $C_i(\sigma) \leq d_j + w_i p_j / w_j$. Therefore, $V(\sigma) - V(\sigma') \geq 0$ either when $w_i \geq w_j$ or when $w_i < w_j$ and $C_i(\sigma) \leq d_j + w_i p_j / w_j$.

It is now clear from the analysis of the cases $C_i(\sigma) \leq d_i + p_j$ and $C_i(\sigma) > d_i + p_j$ that if (b) holds, then $V(\sigma) \geq V(\sigma')$. When $w_i < w_j$ so that (b) is not satisfied, it is apparent from the analysis of the case $C_i(\sigma) \leq d_i + p_j$ that if (c) holds, then $V(\sigma) \geq V(\sigma')$, and from the analysis of the case $C_i(\sigma) > d_i + p_j$ that if (d) holds, then $V(\sigma) \geq V(\sigma')$. □

Apart from (b), the conditions of Lemma 1 depend on the position in σ of the reversed pair ji. However, using Lemma 1 and bounds on $C_i(\sigma)$, Theorem 6 establishes conditions which are independent of the position in σ of the reversed pair ji.

Theorem 6. *If σ is any non-late sequence containing a reversed pair ji, then σ is not EDD-maximal if at least one of the following conditions is satisfied:*
(α) $d_j \geq d_i + p_i - 1$;
(β) $w_i \geq w_j$;
(γ) $p_j \geq p_i - 1$ and $w_j(d_j - d_i) \geq p_j(w_j - w_i)$;
(δ) $w_j(d_j - d_i) \geq \max\{p_j(w_j - w_i), w_j(p_i - 1) - w_i p_j\}$.

Proof. As σ contains only early and partially early jobs, each having an integer processing time and due date, we have

$$C_i(\sigma) \leq d_i + p_i - 1 \tag{7}$$

since otherwise job i would be late. If (α) holds, then (7) yields $C_i(\sigma) \leq d_j$, so the result follows from condition (a) of Lemma 1. Also if (β) holds, then the result follows trivially from condition (b) of Lemma 1.

In the remainder of the proof we assume that (β) does not hold, so $w_i < w_j$. Suppose next that (γ) holds. Substituting $p_i - 1 \leq p_j$ into (7) yields

$$C_i(\sigma) \leq d_i + p_j. \tag{8}$$

Also, the inequality

$$p_j(w_j - w_i) \leq w_j(d_j - d_i) \tag{9}$$

of (γ) implies that

$$d_i + p_j \leq (w_j d_j - w_i d_i)/(w_j - w_i) \tag{10}$$

which, when combined with (8), shows that condition (c) of Lemma 1 is satisfied. Thus, we have established the result if condition (γ) holds and (β) does not.

Finally, suppose that condition (δ) is satisfied. As above, the inequality (9), which is deduced from (δ), implies (10). The other inequality of (δ),

$$w_j(p_i - 1) - w_i p_j \leq w_j(d_j - d_i),$$

which, when combined with (7), shows that

$$C_i(\sigma) \leq d_i + p_i - 1 \leq d_j + w_i p_j / w_j. \tag{11}$$

Now if $C_i(\sigma) \leq d_i + p_j$, then (10) shows that condition (c) of Lemma 1 holds. Alternatively, if $C_i(\sigma) > d_i + p_j$, then (11) shows that condition (d) of Lemma 1 is satisfied. Thus, in both cases at least one condition of Lemma 1 holds to give the required result. □

Next, we establish a lower bound on the completion time of job i of a reversed pair ji in an EDD-maximal sequence.

Theorem 7. *If σ is an EDD-maximal non-late sequence containing a reversed pair ji, then*

$$C_i(\sigma) \geq \min\{1 + \lfloor (w_j d_j - w_i d_i)/(w_j - w_i) \rfloor, 1 + \lfloor d_j + w_i p_j / w_j \rfloor\}. \tag{12}$$

Furthermore, if $p_j \geq p_i - 1$, then

$$C_i(\sigma) \geq 1 + \lfloor (w_j d_j - w_i d_i)/(w_j - w_i) \rfloor. \tag{13}$$

Proof. Since σ is an EDD-maximal sequence, none of the conditions of Lemma 1 are satisfied. If $C_i(\sigma) \leq d_i + p_j$, then because (c) does not hold, we have

$$C_i(\sigma) > (w_j d_j - w_i d_i)/(w_j - w_i). \tag{14}$$

Similarly, if $C_i(\sigma) > d_i + p_j$, the violation of (d) yields

$$C_i(\sigma) > d_j + w_i p_j / w_j. \tag{15}$$

Since $C_i(\sigma) \leq d_i + p_j$ or $C_i(\sigma) > d_i + p_j$, either (14) or (15) holds. Using the integrality of $C_i(\sigma)$ and the lower bound given by the smaller of the right hand sides of (14) and (15), we deduce inequality (12).

We now establish inequality (13) when $p_j \geq p_i - 1$. Combining $C_i(\sigma) \leq d_i + p_i - 1$, which holds because job i is non-late, with $p_i - 1 \leq p_j$ yields $C_i(\sigma) \leq d_i + p_j$. Thus, (14) holds in this case, from which we use the integrality of $C_i(\sigma)$ to obtain (13). □

We explain next how Theorems 6 and 7 are used in a *reversed pair elimination procedure*. Initially, each possible choice of job j and job i, where $d_j > d_i$ and $d_i > p_j$ (if the latter inequality is not satisfied, job i is late when sequenced after job j), defines a candidate reversed pair ji in an optimal EDD-maximal non-late sequence. Many candidates are eliminated immediately because one of the conditions of Theorem 6 holds. For candidates ji which are not eliminated, Theorem 7

defines a lower bound of the form $C_i \geq a_{ji}$ on the completion time of job i if ji is a reversed pair of an EDD-maximal non-late sequence. Let LB_{ji} be a lower bound on the total weighted late work for the constrained problem in which ji is forced to be a reversed pair, i.e., job i is constrained to be sequenced immediately after job j and have a completion time which satisfies $a_{ji} \leq C_i \leq d_i + p_i - 1$. If ji forms a reversed pair in an optimal sequence, then $\mathrm{LB}_{ji} \leq \mathrm{UB}$, where UB is obtained by applying our dynamic programming heuristic of Section 4 (although any other upper bound on the total weighted late work can be used). Thus, if $\mathrm{LB}_{ji} > \mathrm{UB}$, the reversed pair ji cannot exist in an optimal non-late sequence and, hence, is eliminated as a candidate.

It remains to define the lower bound LB_{ji} on the total weighted late work for the constrained problem which is used in the reversed pair elimination procedure. We adopt a similar approach to that for obtaining lower bounds for use in the lateness test. Since job j is constrained to be early and job i is constrained to be completed not after time $d_i + p_i - 1$ (and not before time a_{ji}), we enforce these constraints by assigning large weights to jobs j and i and resetting the due date of job i to $d_i + p_i - 1$. Thus, to obtain LB_{ji}, we solve the preemptive total weighted late work problem having processing times $p'_h = p_h$ $(h = 1, \ldots, n)$, weights $w'_h = w_h$ $(h = 1, \ldots, n;\ h \neq i;\ h \neq j)$ and $w'_i = w'_j = \infty$ and due dates $d'_h = d_h$ $(h = 1, \ldots, n;\ h \neq i)$ and $d'_i = d_i + p_i - 1$. If LB'_{ji} is the total weighted late work for this preemptive problem, obtained by applying the Preemptive Scheduling Algorithm of Section 1, our lower bound is defined by $\mathrm{LB}_{ji} = \mathrm{LB}'_{ji} + w_i(a_{ji} - d_i)$: since the late work for job i is zero in the preemptive problem, but is at least $a_{ji} - d_i$ in the constrained problem, the final term in LB_{ji} accounts for this difference. We now give a formal justification the use of LB_{ji} as a lower bound on the total weighted late work for the (non-preemptive) constrained problem.

Theorem 8. *No optimal EDD-maximal solution contains a reversed pair which is eliminated by the reversed pair elimination procedure.*

Proof. It is sufficient to establish that LB_{ji} is a lower bound on the total weighted late work for the constrained problem. Let σ denote an optimal non-late sequence for the constrained problem which defines $V_h(\sigma)$ as the late work for job h $(h = 1, \ldots, n)$, where $V_h(\sigma) = p_h$ if h does not appear in σ. The constraint $a_{ji} \leq C_i \leq d_i + p_i - 1$ and the observation that $a_{ji} > d_i$ implies job i is partially early in σ. Since job j is early in σ, we obtain

$$w_i V_i(\sigma) + w_j V_j(\sigma) = w_i V_i(\sigma) = w_i(C_i(\sigma) - d_i) \geq w_i(a_{ji} - d_i). \tag{16}$$

Suppose that σ is evaluated with respect to the data for the preemptive problem to give $V'_h(\sigma)$ as the late work for job h $(h = 1, \ldots, n)$. We have $V'_i(\sigma) = V'_j(\sigma) = 0$ since jobs i and j are early with respect to the due dates d'_i and d'_j, whereas $V'_h(\sigma) = V_h(\sigma)$ for $h = 1, \ldots, n$, $h \neq i$ and $h \neq j$. Therefore, we deduce from (16) that

$$\sum_{h=1}^{n} w_h V_h(\sigma) \geq \sum_{h=1}^{n} w_h V'_h(\sigma) + w_i(a_{ji} - d_i) \geq \mathrm{LB}'_{ji} + w_i(a_{ji} - d_i) = \mathrm{LB}_{ji},$$

where the final inequality holds because LB'_{ji} is the minimum total weighted late work for the preemptive problem. It is now apparent that LB_{ji} is a valid lower bound for use in our reversed pair elimination procedure. □

In view of the above analysis, we modify each set A_{jt} so that it contains only those jobs i such that ji remains a candidate after the reversed pair elimination procedure is applied and such that $a_{ji} \leq t \leq d_i + p_i - 1$. The modified A_{jt} is used in the computation of our lower bounds and increases their effectiveness. Also, these modified sets can be used in the dynamic programming algorithm of Section 3.

7. The branch and bound algorithm

In this section, we give full details of our branch and bound algorithm. Prior to applying branch and bound, the earliness test of Section 5.1 is first applied. Then, we use the dynamic programming heuristic of Section 4 (incorporating the redundant state elimination procedure of Section 5.3) to generate an upper bound. To economize on storage space, the recursion is used to obtain the value UB only, with no attempt made to find a corresponding sequence of jobs. (The branch and bound algorithm finds a sequence with total weighted late work of UB or less.) Having obtained UB, the lateness test of Section 5.2 is applied to reduce further the number of jobs. The final preprocessing step applies the reversed pair elimination procedure of Section 6.

The branch and bound algorithm first computes the lower bound of Section 4 (using redundant state elimination and the modified sets A_{jt}), and the corresponding non-late 'sequence' of jobs is obtained. If there are no repeated jobs in this non-late 'sequence', then it provides an optimal solution at the root node of the search tree. Otherwise, a repeated job i is found in the non-late 'sequence'. A binary branching rule constrains job i so that either it is sequenced in EDD order or it is not in EDD order. When job i is sequenced in EDD order, i is removed from each set A_{jt} and the recursion equations for $f'_i(t)$ are modified from those given in Section 4 by deleting the expression $f'_{i-1}(t) + w_i p_i$ which corresponds to cases that job i is late or is deferred. Alternatively, when job i is not sequenced in EDD order (it is either late or deferred), the values of $f'_i(t)$ are computed using $f'_i(t) = f'_{i-1}(t) + w_i p_i$. In either case, job i cannot again appear twice in a 'sequence' obtained from the lower bounding procedure.

For any node of the search tree in which the lower bound is not greater than the current upper bound and the corresponding non-late 'sequence' is feasible, the upper bound is updated and the node is discarded. A node is also discarded if its lower bound exceeds the current upper bound. A newest active node search selects a node from which to branch.

8. Computational experience

The algorithm was tested on problems with numbers of jobs ranging from

$n = 100$ to $n = 700$ in steps of 100. For each job i, an integer processing time p_i was generated from the uniform distribution $[1, 100]$ and an integer weight w_i was generated from the uniform distribution $[1, 10]$. 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, where $d^l \in \{0.2, 0.4, 0.6, 0.8\}$, $d^u \in \{0.4, 0.6, 0.8, 1.0\}$ and $d^l < d^u$ and having computed $P = \sum_{i=1}^{n} p_i$, an integer due date d_i was generated from the uniform distribution $[Pd^l, Pd^u]$ for each job i. For each value of n, five problems were generated for each of the 10 pairs of values of d^l and d^u. This yields 50 problems for each value of n.

Our algorithm was coded in FORTRAN V and run on a CDC 7600 computer. Computational results given in Table 1 aim to demonstrate the accuracy of the upper bound obtained from the dynamic programming heuristic, to assess the effectiveness of the earliness and lateness tests and the reversed pair elimination procedure, to gauge the ability of the lower bounding scheme to provide an optimal solution at the root node of the search tree and to examine the overall effectiveness of the algorithm through average computation times in seconds.

We first observe from Table 1 that our dynamic programming heuristic generates the minimum total weighted late work as an upper bound for all but 8 of the 350 test problems. Since the minimum total weighted late work can be achieved only when there are no deferred jobs in an optimal solution, it is apparent that most test problems have an optimal schedule in which non-late jobs are sequenced in EDD order.

The accuracy of our upper bounds suggests that the lateness test and the reversed pair elimination procedure, which use the value UB, are likely to be effective. Examination of the average number of jobs remaining after the earliness and lateness test are applied, shows that approximately 20% of jobs are eliminated by these tests. A further study of results shows that almost all of these jobs are eliminated using the lateness test. Since the parameter d^u roughly measures the average number of late jobs and for our test problems the average value of d^u is 0.8, the lateness test eliminates almost as many jobs as is possible for a test of this type.

It is also evident from Table 1 that the reversed pair elimination procedure leaves very few candidates to be considered. Since it is only candidate reversed pairs which cause repeated jobs in the non-late 'sequence' corresponding to the lower bound, this non-late 'sequence' is expected to be feasible or almost feasible. Table 1 verifies our intuition, since over 90% of problems are solved at the root node of the search tree with the lower bounding scheme generating a feasible non-late 'sequence'. For 28 of the 29 problems in which this non-late 'sequence' is not feasible, a single branching which creates two search tree nodes in addition to the root node solves the problem: for the other problem, four nodes are created in addition to the root node.

The final column of Table 1 lists average computation times in seconds. Although computation times become quite large for $n = 600$ and $n = 700$, our algorithm, nevertheless, provides an effective method of solution. Its success is attributed to the accurate upper bounds generated by our dynamic programming heuristic, to the ability of the lateness test to eliminate jobs from the problem and

Table 1

Computational results

n	NHO	ANJR	ANRP	NSRN	ACT
100	46	73	2.5	47	0.91
200	49	160	2.7	45	3.44
300	50	246	2.2	45	7.84
400	49	325	2.2	46	12.98
500	49	387	1.9	44	20.71
600	49	487	1.8	49	30.27
700	50	555	4.4	45	43.16

NHO: number of problems (out of 50) for which the dynamic programming heuristic is optimal.

ANJR: average number of jobs remaining after the reduction tests are applied.

ANRP: average number of candidate reversed pairs remaining after the reversed pair elimination procedure is applied.

NSRN: number of problems (out of 50) solved at the root node of the search tree.

ACT: average computation time in seconds.

to the lower bounding procedure which, through the aid of the reversed pair elimination procedure, enables an optimal solution to be generated at the root node of the search tree in many cases.

9. Concluding remarks

We have established the computational complexity of preemptive and non-preemptive scheduling on a single machine to minimize total weighted late work. The preemptive problem is solvable in $O(n \log n)$ time using our algorithm of Section 1. Also, through the use of Theorem 3 which shows that early and partially early jobs are sequenced almost in EDD order, we have derived a dynamic programming algorithm for the non-preemptive problem which requires pseudopolynomial time. Since the non-preemptive total late work problem is already known to be NP-hard, there is little hope of finding a polynomial algorithm.

This paper also gives a practical branch and bound algorithm for the non-preemptive total weighted late work problem. The lateness test of Section 5 and the reversed pair elimination procedure of Section 6 have a major influence on the success of the algorithm. Armed with these devices to restrict the search, our lower bounding scheme, derived using the dynamic programming state-space relaxation method, produces an optimal sequence in many cases. Even if it does not solve the problem at the root node, the branch and bound search tree is likely to be small since all our test problems are solved after at most two branchings.

Acknowledgement

The research by the first author was supported by a grant from King Abdul-Aziz University, Jeddah, Saudi Arabia. The authors are grateful to an anonymous referee for suggestions on improving the structure of the paper.

References

[1] T.S. Abdul-Razaq and C.N. Potts, 1988. *Dynamic Programming State-Space Relaxation for Single Machine Scheduling.* **Journal of the Operational Research Society 39,** 141–152.

[2] J. Blazewicz, 1984. *Scheduling Preemptible Tasks on Parallel Processors with Information Loss,* **Technique et Science Informatique 3,** 415–420.

[3] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest and R.E. Tarjan, 1973. *Time Bounds for Selection,* **Journal of Computer and System Sciences 7,** 448–461.

[4] N. Christofides, A. Mingozzi and P. Toth, 1981. *State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems,* **Networks 11,** 145–164.

[5] R.M. Karp, 1972. *Reducibility among Combinatorial Problems,* in **Complexity of Computer Computations,** J.W. Thatcher and J.W. Miller (eds.), Plenum Press, New York, pp. 85–103.

[6] E.L. Lawler, 1976. *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York.

[7] E.L. Lawler, 1977. *A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness,* **Annals of Discrete Mathematics 1,** 331–342.

[8] E.L. Lawler, and J.M. Moore, 1969. *A Functional Equation and its Application to Resource Allocation and Sequencing Problems,* **Management Science 16,** 77–84.

[9] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, 1977. *Complexity of Machine Scheduling Problems,* **Annals of Discrete Mathematics 1,** 343–362.

[10] C.N. Potts and L.N. Van Wassenhove, 1985. *A Branch and Bound Algorithm for the Total Weighted Tardiness Problem,* **Operations Research 33,** 363–377.

[11] C.N. Potts and L.N. Van Wassenhove, 1988. *Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs,* **Management Science 34,** 843–858.

[12] C.N. Potts and L.N. Van Wassenhove, 1992. *Single Machine Scheduling to Minimize Total Late Work,* **Operations Research 40,** to appear.

[13] A. Schonhage, M. Paterson and M. Pippenger, 1976. *Finding the Median,* **Journal of Computer and System Sciences 13,** 189–199.