Scheduling a Single Machine to Minimize the Number of Late Jobs

E. L. Lawler

Computer Science Division
University of California at Berkeley

ABSTRACT

Suppose n jobs, each with a specified release date, due date and processing time, are to be scheduled for processing by a single machine, with the objective of minimizing the number of jobs that are not completed by their due dates. Three results, in the form of two algorithms and one NP-completeness proof, are presented. Our first result is to show that if release dates and due dates are "compatible," i. e. similarly ordered, an optimal schedule can be found in $O(n \log n)$ time by a procedure that is a natural generalization of that of Moore for the case in which all release dates are equal. This result improves on an $O(n^2)$ solution to this problem by Kise, Ibaraki and Mine. For our second result, no particular relationship is assumed between the release dates and the due dates and the schedule is allowed to be preemptive. We show that a schedule minimizing the weighted number of late jobs can be found in $O(n^3 W^3)$ time where W is the sum of the integer weights assigned to the jobs. If the jobs are unweighted, then in effect $W = n$ and the time bound reduces to $O(n^6)$, thereby yielding a polynomial-bounded algorithm for a problem for which no such solution procedure was previously known. For our third result, we suppose that all release dates are equal and that each job has a deadline in addition to its due date. We show that it is an NP-hard problem to minimize the number of late jobs (with respect to due dates) while observing all deadlines. This result resolves an open question suggested by a result of J. Sidney.

Keywords and phrases: scheduling, single machine, release dates, due dates, deadlines, preemption, dynamic programming, pseudo-polynomial algorithm, polynomial algorithm, NP-completeness.

This research was supported by NSF Grant MCS78-20054, administered by the Electronics Research Laboratory, the University of California at Berkeley.

1. INTRODUCTION

There are n *jobs*, $j = 1,2,...,n$, each with a specified *release date* r_j, *due date* d_j and *processing time* p_j, to be scheduled for processing by a single machine that can work on at most one job at a time. The processing of job j cannot begin before its release date r_j. If preemption is not permitted, job j must be processed continuously for time p_j. If preemption is permitted, the processing of job j may be interrupted at any time and resumed at a later time, provided that the total amount of time that j is processed is equal to p_j. If job j is completed at or before its due date, it is said to be *on time*; otherwise it is *late*.

In addition to the values r_j, d_j and p_j, an integer *weight* w_j may be specified for each job. In this case, our objective is to minimize the sum of the weights of the late jobs. Otherwise, our objective is simply to minimize the number of late jobs. (The unweighted case is of course equivalent to $w_j = 1$ for all j.)

We summarize below previous results which have been obtained for variations of this general problem. In each case, we indicate parenthetically the designation of the problem in the notation of [6].

(*i*) $(1\|\Sigma U_j)$ *Jobs unweighted, all release dates equal.* (Without loss of generality let $r_j = 0$ for all j.) Moore [8] showed that this problem can be solved in $O(n \log n)$ time. Sidney [9] showed that Moore's algorithm can be extended to the case in which a specified subset of the jobs are required to be on time. Lawler [4] showed that Moore's algorithm can be extended to the weighted case, provided weights and processing times are *agreeable*, i. e. the jobs can be indexed so that $p_1 \le p_2 \le \ldots \le p_n$ and $w_1 \ge w_2 \ge \ldots \ge w_n$.

(*ii*) $(1|r_j|\Sigma U_j)$ *Jobs unweighted, unequal release dates.* This problem is NP-complete in the strong sense, i.e. with respect to unary encoding of data. However, Kise, Ibaraki and Mine [2] showed that this problem can be solved in $O(n^2)$ time, provided release dates and due dates are *compatible*, i. e. the jobs can be indexed so that $r_1 \le r_2 \le \ldots \le r_n$ and $d_1 \le d_2 \le \ldots \le d_n$.

(*iii*) $(1\|\Sigma w_j U_j)$ *Jobs weighted, release dates equal.* This problem is NP-complete (in the ordinary sense), since there is an immediate reduction from the (0, 1)-knapsack problem. However, Lawler and Moore [3] showed that this problem can be solved within the pseudopolynomial time bound of $O(nW)$, where W is the sum of the job weights, by an extension of the well-known dynamic programming computation for knapsack problems. Moreover, there is no difficulty in generalizing this result to the case in which release dates and due dates are unequal but compatible, in the same sense as [2].

There is no advantage to preemption if all release dates are equal or, more generally, if release dates and due dates are compatible. That is, there exists an optimal nonpreemptive schedule as good as any preemptive one. Hence there is no point in considering the possibility of preemption in (*i*) and (*iii*) above. However, the possibility of preemption does make a very significant difference if release dates are not compatible with due dates.

Our first result is to show that the special case of $1|r_j|\Sigma U_j$ dealt with by Kise, et al can be solved in $O(n \log n)$ time by an algorithm that is a natural generalization of that of Moore. This algorithm can be easily modified to deal with the generalization of Sidney, but it does not appear to be extendible to the case of agreeable weights.

Our second result is to show that there is a pseudopolynomial algorithm for the case of weighted jobs and arbitrary release dates, if preemption is permitted $(1|pmtn, r_j|\Sigma w_j U_j)$. The time bound of this algorithm is $O(n^3 W^3)$, which means that it is truly polynomial bounded, $O(n^6)$, if jobs are unweighted. It is possible that this algorithm can be modified to accommodate the constraints of Sidney's generalization, but we do not attempt to do this here.

There is a natural further generalization of the constraints of Sidney, as follows. In addition to a due date d_j for each job j, a *deadline* $\bar{d}_j$ is specified, with $d_j \leq \bar{d}_j$. A job may be completed later than its due date d_j (it is then simply "late") but under no circumstances may it fail to meet its deadline (the schedule is then infeasible). Sidney's case is then simply that in which either $\bar{d}_j = d_j$ or $\bar{d}_j + \infty$. Our third result is to show that with the addition of deadlines, the problem becomes NP-hard in the ordinary sense. It is not known whether this problem is NP-hard in the strong sense or whether it admits of a pseudopolynomial algorithm.

For the purpose of comparison, we mention some results concerning the scheduling of parallel machines with the objective of minimizing the number of late jobs. For two identical parallel machines, the nonpreemptive scheduling problem is NP-complete in the ordinary sense, even if all release dates are equal. (This is $P2\|\Sigma U_j$, in the notation of [6].) For an arbitrary number of m of identical parallel machines (where m is specified as part of the problem instance) the problem $(P\|\Sigma U_j)$ is NP-complete in the strong sense. These results are obtained by simple transformation from PARTITION and 3-PARTITION respectively. If preemption is permitted and the number of identical parallel machines is arbitrary, the problem $(P|pmtn|\Sigma U_j)$ is NP-complete in the ordinary sense, as can be shown by transformation from PARTITION [7]. However, if there is a fixed number m of "uniform" machines (i. e. machines of different speeds) and the jobs are weighted, the problem (denoted $Qm|pmtn|\Sigma w_j U_j$) can be solved in pseudopolynomial time ($O(n^2 W^2)$ if $m = 2$ and $O(n^{3m-5} W^2)$ if $m \geq 3$) [5]. If the jobs are unweighted, these pseudopolynomial time bounds become polynomial ($O(n^4)$ and $O(n^{3(m-1)})$ respectively).

2. COMPATIBLE RELEASE DATES AND DUE DATES

Let us consider the problem of minimizing the (unweighted) number of late jobs in the case in which release dates and due dates are compatible. We assume release dates are nonnegative and that the jobs are indexed so that $r_1 \leq r_2 \leq \ldots \leq r_n$ and $d_1 \leq d_2 \leq \ldots \leq d_n$. (If the jobs are not so indexed, this can be done in $O(n \log n)$ time by sorting.)

Call a subset $S \subseteq N = \{1,2,\ldots,n\}$ *feasible* if there exists a schedule for S in which all jobs are on time. It is simple to determine whether or not a given subset S is feasible by invoking the EDD (earliest due date) scheduling rule: Schedule the jobs in order of their indices, with each job being performed as early as possible (subject to its release date and the completion of the previous job). S is feasible if and only if all jobs in S are on time when scheduled in this way.

Observe that if a job is late, it makes no difference how long its processing is postponed. Hence an optimal schedule can be assumed to consist of a feasible subset of jobs S that are processed in EDD order, followed by the complementary set of jobs $N-S$ that are processed in any order. Our problem thus reduces to that of finding a maximum feasible set $S \subseteq N$.

Let $C^{(j)}(k)$ denote the earliest completion time for any feasible subset of size k contained in $\{1,2,\ldots,j\}$. (If no feasible subset of size k exists, let $C^{(j)}(k) = +\infty$.) We define

$$C^{(0)}(k) = \begin{cases} 0 & \text{if } k = 0 \\ +\infty & \text{otherwise} \end{cases}$$

For $j = 1,2,\ldots,n$ we have the recurrence relations

$$C^{(j+1)}(k) = \begin{cases} C^{(j)}(k) & \text{if } max\{r_{j+1}, C^{(j)}(k-1)\} + p_{j+1} > d_{j+1} \\ min\{C^{(j)}(k), max\{r_{j+1}, C^{(j)}(k-1)\} + p_{j+1}\} & \text{otherwise} \end{cases} \quad (2.1)$$

Equations (2.1) are easily justified and should require no explanation. These equations can be solved, for all j and k, in $O(n^2)$ time. A maximum feasible subset of N contains k^* jobs, where k^* is the largest value of k such that $C^{(n)}(k)$ is finite.

Note that recurrence relations (2.1) are easily modified to solve the problem for weighted jobs in $O(nW)$ time. Let $C^{(j)}(w)$ denote the earliest possible completion time for a subset of weight w, and replace $C^{(j+1)}(k)$, $C^{(j)}(k)$, $C^{(j)}(k-1)$ by $C^{(j+1)}(w)$, $C^{(j)}(w)$, $C^{(j)}(w - w_{j+1})$ in (2.1). The resulting equations are effectively a generalization of those in [3].

Now let us consider how to reduce the bound on the running time from $O(n^2)$ to $O(n \log n)$. For given j, let k' denote the smallest value of k such that $C^{(j)}(k) > r_j$ and k'' denote the largest value of k such that $C^{(j)}(k)$ is finite.

Lemma 1 For all j, $0 \le j \le n$,

$$C^{(j)}(k'+1) - C^{(j)}(k') \ge C^{(j)}(k') - r_j$$
$$C^{(j)}(k+1) - C^{(j)}(k) \ge C^{(j)}(k) - C^{(j)}(k-1), \quad \text{for } k' < k < k''$$

Proof: By induction. The lemma is true for $j = 1$. So assume it is true for j. Let l be the smallest value of k such that

$$C^{(j)}(k) > max\{r_{j+1}, C^{(j)}(k-1)\} > p_{j+1}.$$

It then follows from (2.1) that

$$C^{(j+1)}(k) = \begin{cases} C^{(j)}(k) & \text{for } k < l \\ max\{r_{j+1}, C^{(j)}(k-1)\} + p_{j+1}, & \text{for } k \ge l \end{cases} \quad (2.2)$$

It is then easy to establish that the lemma is true for $j + 1$.

Lemma 2 For all j, $0 \le j \le n$, there exists a tower of feasible subsets

$$\emptyset = S^{(j)}(0) \subset S^{(j)}(1) \subset \ldots \subset S^{(j)}(k''), \subseteq \{1,2,\ldots,j\},$$

where $S^{(j)}(k)$ contains k jobs, and $S^{(j)}(k)$ is completed at time $C^{(j)}(k)$ when scheduled by the EDD rule. Moreover, if $S^{(j)}(k) - S^{(j)}(k-1) = \{i(k)\}$, for $1 \le k \le k''$, then

$$C^{(j)}(k') - \tau_j \le p_{i(k')}$$
$$C^{(j)}(k) - C^{(j)}(k-1) = p_{i(k)}, \text{ for } k' < k \le k''.$$

Proof: By induction, similar to that in the proof of Lemma 1. From (2.2),

$$S^{(j+1)}(k) = \begin{cases} S^{(j)}(k) & \text{for } k < l \\ S^{(j)}(k-1) \cup \{j+1\} & \text{for } k \ge l \end{cases}$$

It is then easy to establish that the tower exists for $j + 1$. ∎

As before, for given j, let k' denote the smallest value of k such that $C^{(j)}(k) > \tau_j$ and k'' denote the largest value of k such that $C^{(j)}(k)$ is finite. Let

$$S^{(j)} = S^{(j)}(k'') - S^{(j)}(k'-1).$$

and suppose that $p_{i(k')}$ has been reduced in value to $C^{(j)}(k') - \tau_j$. We shall now show how to construct $S^{(j+1)}$ from $S^{(j)}$.

Let $S^{(j)} = \{i(k'), i(k'+1), \ldots, i(k'')\}$, where

$$p_{i(k')} \le p_{i(k'+1)} \le \ldots \le p_{i(k'')}.$$

Note that

$$C^{(j)}(k) = \tau_j + p_{i(k')} + \ldots + p_{i(k)}, \quad \text{for } k' \le k \le k''.$$

So if there is an l', $k' \le l' < k''$ such that

$$p_{i(k')} + \ldots + p_{i(l'-1)} \le \tau_{j+1} - \tau_j < p_{i(k')} + \ldots + p_{i(l')},$$

then we reduce $p_{i(l')}$ to

$$p_{i(l')} := p_{i(k')} + \ldots + p_{i(l')} - (\tau_{j+1} - \tau_j),$$

and we shall have

$$C^{(j)}(k) = \tau_{j+1} + p_{i(l')} + \ldots + p_{i(k)}, \quad \text{for } l' \le k < k''.$$

Accordingly, we remove jobs from $S^{(j)}$ in nondecreasing order of procesing times until either $S^{(j)}$ is emptied or until job $i(l')$ is identified. The processing time of job $i(l')$ is reduced as necessary. (Jobs $i(k'),\ldots,i(l'-1)$ will be contained in the maximum feasible set we shall construct, and we store them away in a set we designate S'.)

For notational convenience, let $p(S) = \Sigma_{i \in S} p_i$, for any set $S \subseteq N$. If now

$$r_{j+1} + p(S^{(j)} \cup \{j+1\}) \leq d_{j+1},$$

we are done; $S^{(j+1)} = S^{(j)} \cup \{j+1\}$. Otherwise, we remove from $S^{(j)} \cup \{j+1\}$ a job with maximum processing time to obtain $S^{(j+1)}$.

Consider a small example with seven jobs:

j	r_j	d_j	p_j
1	0	4	3
2	2	4	2
3	4	9	4
4	5	9	1
5	6	12	3
6	7	12	2
7	8	13	1

The algorithm proceeds as follows.

Iteration 1: $S^{(1)} = \{1\}$, $S' = \emptyset$.

Iteration 2: Since $r_2 - r_1 = 2$, p_1 is reduced from 3 to 1. Since $r_2 + p(S^{(1)} \cup \{2\}) > d_2$, and $p_2 = max\{p_i | i \in S^{(1)} \cup \{2\}\}$, we have $S^{(2)} = \{1\}$ and $S' = \emptyset$.

Iteration 3: Since $p_1 = 1 < r_3 - r_2 = 2$, we remove job 1 from $S^{(2)}$ and place it in S'. We now have $S^{(2)} = \emptyset$, $S' = \{1\}$. Since $r_3 + p(S^{(2)} \cup \{3\}) \leq d_3$, we have $S^{(3)} = \{3\}$.

Iteration 4: Since $r_4 - r_3 = 1 < p_3 = 4$, we reduce p_3 to 3. Since $r_4 + p(S^{(3)} \cup \{4\}) \leq d_4$, we now have $S^{(4)} = \{3, 4\}$, $S' = \{1\}$.

Iteration 5: Since $r_5 - r_4 = 1 = p_4 = min\{p_i | i \in S^{(5)}\}$, we remove job 4 from $S^{(4)}$, giving $S^{(4)} = \{3\}$, $S' = \{1, 4\}$. Now $r_5 + p(S^{(4)} \cup \{5\}) = d_5$, so we have $S^{(5)} = \{3, 5\}$.

Iteration 6: Since $r_6 - r_5 = 1 < p_3 = min\{p_i | i \in S^{(6)}\}$, we reduce p_3 from 3 to 2. (We could equally well reduce p_5.) Now $r_6 + p(S^{(5)} \cup \{6\}) > d_6$, and $p_5 = max\{p_i | i \in S^{(5)} \cup \{6\}$, so we throw job 5 away yielding $S^{(6)} = \{3, 6\}$, $S' = \{1, 4\}$.

Iteration 7: Since $r_7 - r_6 = 1 < p_3 = 2$, we reduce p_3 to 1. We now have $r_7 + p(S^{(6)} \cup \{7\}) < d_7$, so we have $S^{(7)} = \{3, 6, 7\}$, with $S' = \{1, 4\}$.

It thus follows that a maximum feasible subset for our example is $S^{(7)} \cup S' = \{1, 3, 4, 6, 7\}$. A feasible schedule for this set is indicated in Figure 1.

An Algol-like program for the algorithm is given below. Notational conventions are as follows: The superscript is dropped from $S^{(j)}$. The variable p denotes $p(S^{(j)})$. We assume $r_o = 0$. We let "$arg\ min\ \{p_h | h \in S\}$" denote the index i such that $p_i = min\ \{p_h | h \in S\}$.

```
begin
   S := ∅;
   S' := ∅;
   p := 0;
   for j := 1 to n
   do τ := r_j − r_{j−1};
      while p > 0 and τ > 0
      do i := arg min {p_h | h ∈ S};
         if τ ≥ p_i then
            S := S − {i};
            S' := S' ∪ {i};
            p := p − p_i;
            τ := τ − p_i
         else
            p_i := p_i − τ;
            p := p − τ;
            τ := 0
         fi
      od
      S := S ∪ {j};
      p := p + p_j;
      if r_j + p > d_j then
         i := arg max {p_h | h ∈ S};
         S := S − {i};
         p := p − p_i
      fi
   od
end
```

Note that initial sorting of jobs by release dates and due dates can be carried out in $O(n\ log\ n)$ time. The maximum or minimum of S must be found no more than $O(n)$ times. It is a straightforward matter to devise a data structure which supports the operations *MIN*, *MAX*, *INSERT*, and *DELETE* with at most $O(log\ n)$ time per operation. Accordingly, the overall running time is bounded by $O(n\ log\ n)$.

Note that if release dates are equal (and can be assumed to be zero), the **while** loop is inoperative and the procedure reduces to that of Moore [4, 8].

Finally as in Sidney [9], suppose that a certain subset of jobs $T \subseteq N$ are required to be completed on time. This constraint can be dealt with as follows. Assume T is a feasible set. (If not, no feasible schedule exists). Then replace "**if** $r_j + p > d_j$ **then**..." with the **while** loop below:

```
while   r_j + p > d_j
do      i := arg max {p_h | h ∈ S − T};
        S := S − {i};
        p := p − p_i
od
```

Justification is left for the reader.

3. PREEMPTIVE SCHEDULING WITH ARBITRARY RELEASE DATES

We now consider the problem of minimizing the weighted number of jobs, when preemption is permitted. We assume no particular relationship between release dales and due dates, but assume the jobs have been numbered in order of due dates, $d_1 \le d_2 \le \ldots \le d_n$. Let $W = \sum_j w_j$.

Our task is to find a feasible set $S \subseteq N$ for which the sum of the weights of the jobs is maximized. As in the previous section, it is simple to determine whether or not a subset S is feasible. The EDD rule is applied preemptively: At each successive point in time, process the available job with least index. (A job is "available" at time t if $t \ge r_j$ and its processing has not yet been completed.) The resulting schedule naturally decomposes into *blocks*, where a block $B \subseteq S$ is defined as a minimal set of jobs without idle time from $r(B) = min_{j \in B}\{r_j\}$ until $t(B) = r(B) + p(B)$, such that each job $j \notin B$ is either completed not later than $r(B)$ or not released before $t(B)$. For this notion of block, see also [1].

Let $C^{(j)}(w,r)$ denote the earliest possible completion time for a feasible subset $B \subseteq \{1,2,\ldots,j\}$ such that

(*i*) $\sum_{k \in B} w_k = w$
(*ii*) $r(B) = r$
(*iii*) B is scheduled as a block by the EDD rule.

Let $P^{(j)}(w,r,t)$, where r is a release date and t is either a release date or $+\infty$, denote the minimum total processing time for a feasible subset $S \subseteq \{1,2,\ldots,j\}$ such that

(*i*) $\sum_{k \in S} w_k = w$
(*ii*) $r(S) \ge r$
(*iii*) all jobs in S are completed by time t when scheduled by the EDD rule.

Let

$$C^{(1)}(w,r) = \begin{cases} r & \text{if } w = 0 \\ r_1 + p_1 & \text{if } w = w_1,\ r = r_1 \\ +\infty & \text{otherwise} \end{cases}$$

and

$$P^{(1)}(w,r,t) = \begin{cases} 0 & \text{if } w = 0 \\ p_1 & \text{if } w = w_1,\ r \le r_1,\ t \ge r_1 + p_1 \\ +\infty & \text{otherwise} \end{cases}$$

Now suppose we are given $C^{(j)}(w,\tau)$ and $P^{(j)}(w,\tau,t)$ for all w,τ,t. How do we compute $C^{(j+1)}(w,\tau)$ and $P^{(j+1)}(w,\tau,t)$?

First consider the computation of $C^{(j+1)}(w,\tau)$. Let $B \subseteq \{1,2,\ldots,j+1\}$ be a block of weight w such that $r(B) = \tau$ and $t(B) = C^{(j+1)}(w,\tau)$. We distinguish three cases:

Case 1: $j+1 \notin B$, in which case $C^{(j+1)}(w,\tau) = C^{(j)}(w,\tau)$.

Case 2: $j+1 \in B$, $\tau \leq \tau_{j+1}$ and job $j+1$ is processed continuously after a (possibly empty) subblock $B_0 \subseteq \{1,2,\ldots,j\}$, as shown in the upper part of Figure 2. (The shaded portions of the figure indicate processing of job $j+1$.) The weight of B_0 is $w - w_{j+1}$ and B_0 finishes at time $C^{(j)}(w - w_{j+1}, \tau) \geq \tau_{j+1}$ (with strict inequality if $w - w_{j+1} > 0$), else B is not an optimal block. In this case,

$$C^{(j+1)}(w,\tau) = C^{(j)}(w - w_{j+1},\tau) + p_{j+1} \leq d_{j+1}$$

Case 3: $j+1 \in B$, $\tau \leq \tau_{j+1}$ and job $j+1$ is processed after a (possibly empty) subblock B_0, with preemptions for subblocks $B_1, B_2, \ldots, B_s$, where $s \geq 1$ and $B_i \subseteq \{1,2,\ldots,j\}$, $i = 0,1,\ldots,s$, as shown in the lower part of Figure 2. Let $w' \leq w - w_{j+1}$ be the weight of B_0, $w'' \leq w - w' - w_{j+1}$ be the weight of B_s, and let B_s begin at time τ''. Then B_0 finishes at time $C^{(j)}(w', \tau) \geq \tau_{j+1}$ (with strict inequality if $w' > 0$) and B_s finishes at time $C^{(j)}(w'', \tau'')$, else B is not an optimal block. Subblocks $B_1,\ldots,B_{s-1}$ have total weight $w - w' - w'' - w_{j+1}$ and total processing time $P^{(j)}(w - w' - w'' - w_{j+1}, \tau', \tau'')$, where τ' is the earliest release date not earlier than $C^{(j)}(w',\tau)$, else B is not an optimal block. Moreover, it must be the case that

$$\tau'' - C^{(j)}(w',\tau) - P^{(j)}(\tau - w' - w' - w_{j+1}, \tau', \tau'') < p_{j+1},$$

else B is not a block. It follows that in this case

$$C^{(j+1)}(w,\tau) = C^{(j)}(w',\tau) + P^{(j)}(w - w' - w'' - w_{j+1}, \tau', \tau'') + (C^{(j)}(w'',\tau'') - \tau'') + p_{j+1} \leq d_{j+1}.$$

These observations can be summarized as follows: If $\tau > \tau_{j+1}$, then

$$C^{(j+1)}(w,\tau) = C^{(j)}(w,\tau).$$

If $\tau \leq \tau_{j+1}$,

$$C^{(j+1)}(w,\tau) = \min\{C^{(j)}(w,\tau), C_2, C_3\}. \tag{3.1}$$

where

$$C_2 = \begin{cases} C^{(j)}(w - w_{j+1},\tau) + p_{j+1}, & \text{if } \tau_{j+1} < C^{(j)}(w - w_{j+1},\tau) \leq d_{j+1} - p_{j+1} \\ +\infty, & \text{otherwise} \end{cases}$$

and

$$C_3 = min\ \{C^{(j)}(w',\tau) + P^{(j)}(w - w' - w'' - w_{j+1},\tau',\tau'') + C^{(j)}(w'',\tau'') - \tau''\} + p_{j+1},$$

where the minimization is taken over all combinations of w',w'' and τ'' such that

(i) $w' + w'' + w_{j+1} = w$

(ii) $C^{(j)}(w',\tau) \geq \tau_{j+1}$, with strict equality if $w' > 0$,

(iii) $C^{(j)}(w',\tau) \leq \tau''$, and

(iv) $\tau'' - p_{j+1} < C^{(j)}(w',\tau) + P^{(j)}(w - w' - w'' - w_{j+1},\tau',\tau'')$
$\leq \tau'' - p_{j+1} - C^{(j)}(w'',\tau'') + d_{j+1}$,

and where τ' is the earliest date not earlier than $C^{(j)}(w',\tau)$. If there are no such w',w'' and τ'' then $C_3 = +\infty$.

Now consider the computation of $P^{(j+1)}(w,\tau,t)$, given $C^{(j+1)}(w,\tau)$, $P^{(j)}(w,\tau,t)$, for all w,τ and t. We first note that if $\tau > \tau_{j+1}$ or if $t \leq \tau_{j+1}$ then $P_{(j+1)}(w,\tau,t) = P^{(j)}(w,\tau,t)$. This follows from the fact that if $j+1 \in S$, then the processing of $j+1$ will not occur between τ and t. Moreover, if $w = 0$, we of course have $P^{(j+1)}(w,\tau,t) = P^{(j)}(w,\tau,t) = 0$. So suppose $w > 0$, $\tau \leq \tau_{j+1} < t$. An optimal set S must contain a final block B. Let $\tau' < t$ be the time the processing of this block begins and let w' be its weight. This block is completed at time $C^{(j+1)}(w',\tau') \leq t$, and has total processing time $C^{(j+1)}(w',\tau') - \tau'$, else S is not optimal. Moreover, the blocks preceding B must have total weight $w - w'$ and total processing time $P^{(j+1)}(w - w',\tau,\tau')$, else S is not optimal. It follows that

$$P^{(j+1)}(w,\tau,t) = min\ \{P^{(j+1)}(w - w',\tau,\tau') + C^{(j+1)}(w',\tau')\}, \tag{3.2}$$

where the minimization is taken over all $\tau' < t$, $w' \leq w$ such that $C^{(j+1)}(w',\tau') \leq t$. If the computation is carried out for increasing values of t in an outer loop and increasing values of τ in an inner loop, previously computed values of $P^{(j+1)}$ are available for the right hand side of (3.2).

The solution to the scheduling problem is given by the largest value of w such that $P^{(n)}(w,\tau,+\infty)$ is finite, where $\tau = min\ \{\tau_1, \tau_2,\ldots,\tau_n\}$. (In the accepted tradition of dynamic programming, we content ourselves with the computation of the *value* of an optimal solution, leaving it as an exercise for the reader to note that it is possible to actually construct an optimal solution by these techniques.)

Finally, we consider the computational complexity of this procedure. There are $O(n^2W)$ equations (3.1) to solve, since $j = 1,2,\ldots,n-1$; $0 \leq w \leq W$, $\tau \in \{\tau_1, \tau_2,\ldots,\tau_n\}$. Each equation requires $O(nW^2)$ operations, since $0 \leq w' \leq W$, $0 \leq w'' \leq W$, $\tau'' \in \{\tau_1, \tau_2,\ldots,\tau_n\}$, in the computation of C_3. Hence $O(n^3W^3)$ time is required for equations (3.1). There are $O(n^3W)$ equations (3.2) to solve, since $j = 1,2,\ldots,n-1$; $0 \leq w \leq W$, $\tau \in \{\tau_1,\tau_2,\ldots,\tau_n\}$, $t \in \{\tau_1,\tau_2,\ldots,\tau_n\} \cup \{+\infty\}$. Each equation requires $O(nW)$ operations, since $\tau' \in \{\tau_1,\tau_2,\ldots,\tau_n\}$, $0 \leq w' \leq W$. Hence $O(n^4W^2)$ time is required for equations (3.2). Noting that $W \geq n$, we conclude that the dynamic programming computation requires $O(n^3W^3)$ time and $O(n^3W)$ space. In the unweighted case, with $W = n$, this reduces to $O(n^6)$ time and $O(n^4)$ space.

4. NP-COMPLETENESS OF DEADLINE PROBLEM

We shall now show that the following problem is NP-complete.

SCHEDULING WITH DUE DATES AND DEADLINES

Instance: (i) n triples of positive integers $(p_j, d_j, \bar{d}_j)$ specifying jobs, where d_j is the due date, $\bar{d}_j$ is the deadline, and p_j is the processing time of job j, for $j = 1,2,...,n$. (ii) a positive integer $k \leq n$.

Question: Can the jobs be scheduled for processing by a single machine so that all jobs are completed by their deadlines and at least k of the jobs are *on time*, i.e. completed by their due dates?

Note that SCHEDULING is clearly in NP. We can determine whether or not there exists a feasible schedule in which a specified subset S of jobs is on time by applying the EDD rule: Schedule the jobs in order of the values δ_j, where $\delta_j = d_j$ if $j \in S$ and $\delta_j = \bar{d}_j$ otherwise. There exists a feasible schedule in which each job in S is on time, if and only if each job j is completed by time δ_j in this schedule.

We shall exhibit a polynomial transformation from PARTITION: Given t positive integers $a_1, a_2, ..., a_t$, where $\sum a_i = 2b$, does there exist a subset S such that $\sum_{i \in S} a_i = b$? For a given instance of PARTITION, we shall create an instance of SCHEDULING with $n = 4t$ jobs and $k = 2t$.

As a preliminary, consider an instance of SCHEDULING with $4t$ jobs, which we give the indices $j = i, i + t, i + 2t, i + 3t$, for $i = 1,2,...,t$. For these jobs we specify processing times, due dates, and deadlines as indicated in the table below:

j	p_j	d_j	$\bar{d}_j$
i	2^{i-1}	$2^i - 1$	$P_{i-1} + 2^{i-1}$
$i+t$	2^{i-1}	$2^i - 1$	$P_{i-1} + 2^{i-1} + 2^{t+i-1}$
$i+2t$	2^{t+i-1}	$2^{t+i} - 1$	$P_{i-1} + 2^{t+i-1}$
$i+3t$	2^{t+i-1}	$2^{t+i} - 1$	$P_{i-1} + 2^{i-1} + 2^{t+i-1}$

The deadlines are defined in terms of P_{i-1}, where by definition

$$P_{i-1} = \sum_{k=1}^{t} (2^{t+k-1} + 2^{k-1}) + \sum_{k=1}^{i-1} (2^{t+k-1} + 2^{k-1})$$

$$= 2^t (2^t - 1) + 2^t - 1 + 2^t (2^{i-1} - 1) + 2^{i-1} - 1.$$

Notice that for a given value of i there are two "short" jobs $i, i + t$ and two "long" jobs $i + 2t, i + 3t$. We make the following observations:

(1) There is no feasible schedule in which both i and $i + t$ are on time, since

$$p_i + p_{i+t} = 2^{i-1} + 2^{i-1} = 2^i > 2^i - 1 = d_i = d_{i+t}.$$

(2) Similarly, there is no feasible schedule in which both $i + 2t$ and $i + 3t$ are on time.

(3) Consequently, there is no feasible schedule with more than $2t$ on time jobs.

(4) There is no feasible schedule with $2t$ on time jobs in which both $i + t$ and $i + 3t$ are on time. (If there were such a schedule in which both $i + t$ and $i + 3t$ were on time, then *(i)* either i or $i + 2t$ would fail to meet its deadline or *(ii)* some other job would fail to meet its deadline.)

We are now ready to present the transformation from PARTITION to SCHEDULING. For each number a_i, create four jobs, as indicated in the table below.

j	p_j	d_j	$\bar{d}_j$
i	$2^{i-1}M + a_i$	$(2^i - 1)M + b$	$(P_{i-1} + 2^{i-1})M - b$
$i + t$	$2^{i-1}M$	$(2^i - 1)M + b$	$(P_{i-1} + 2^{i-1} + 2^{t+i-1})M - b$
$i + 2t$	$2^{t+i-1}M$	$(2^{t+i} - 1)M + b$	$(P_{i-1} + 2^{t+i-1})M - b$
$i + 3t$	$2^{t+i-1}M - 2a_i$	$(2^{t+i} - 1)M + b$	$(P_{i-1} + 2^{i-1} + 2^{t+i-1})M - b$

Important exception: For $i = t$, let $d_{i+2t} = d_{i+3t} = (2^{2t} - 1)M - b$. (I.e. *minus* b instead of plus b.)

In the table above, M represents a suitably large number. (It will turn out that we can let $M = 3b$.) Multiplying values $p_j, d_j, \bar{d}_j$ by M does not affect the validity of observations (1) - (4) above. What we must be concerned with is the effect of perturbing the multiplied values by $+ a_i$, $- 2a_i$, $+ b$ and $- b$.

We assert that if there exists a feasible solution to PARTITION then there exists a feasible solution to the corresponding instance of SCHEDULING in which $2t$ jobs are on time. Suppose S is such that $\sum_{i \in S} a_i = b$. Then construct a schedule in which jobs i, $i + 3t$ are on time if $i \in S$ and $i + t$, $i + 2t$ are on time if $i \notin S$. Note that the sum of the processing times for the t jobs where i, $i \in S$, and $i + t$, where $i \notin S$, is $(2^t - 1)M + b$. Moreover these jobs can all be completed on time by scheduling them in due date order at the first of the schedule. The sum of the processing times for the t jobs $i + 3t$, where $i \in S$, and $i + 2t$, where $i \notin S$, is $2^t (2^t - 1)M - 2b$. These jobs can all be completed on time by scheduling them in due date order, after the first t on time jobs. The sum of the processing times for the first $2t$ jobs is then

$$(2^t - 1)M + b + 2^t (2^t - 1)M - 2b = (2^{2t} - 1)M - b,$$

which is precisely the due date $d_{i+2t} = d_{i+3t}$ specified by our exception for the case $i = t$. The remaining $2t$ jobs can all be scheduled, in deadline order, to meet their deadlines.

Now suppose there is a feasible schedule with $2t$ on time jobs. We have already observed in (4) that, because of the structure of the deadlines, jobs $i + t$ and $i + 3t$ cannot both be on time, for any i. We now assert that, because of the perturbation of the due dates by a_i and $-2a_i$, jobs i and $i + 2t$ cannot both be on time. Let S_k denote the set of indices i such that $i + kt$ is on time,

for $k = 0,1,2,3$. Note that

$$|S_0 \cup S_1| = |S_2 \cup S_3| = t,$$

which implies that

$$\sum_{i \in S_0} a_i \leq b,$$

and that

$$\sum_{i \in S_0} a_i - 2 \sum_{i \in S_3} a_i \leq -b,$$

from which

$$\sum_{i \in S_3} a_i \geq b$$

follows. But there is no pair $i + t, i + 3t$ on time, since we have ruled this out in (4). So if there is any pair $i, i + 2t$ on time, then we must have

$$\sum_{i \in S_3} a_i < b,$$

a contradiction. It follows that the only on time pairs are either $i, i + 3t$ or $i + t, i + 2t$, so that $S_0 = S_3$ and we must then have

$$\sum_{i \in S_0} a_i = b.$$

This shows that if there is a feasible schedule with $2t$ on time jobs, then there is a feasible solution to the instance of PARTITION.

References

[1] K. R. Baker, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints," *Oper. Res.*, to appear.

[2] H. Kise, T. Ibaraki, H. Mine, "A solvable case of the one-machine scheduling problem with ready and due times," *Oper. Res.*, 26 (1978) 121-126.

[3] E. L. Lawler, J. M. Moore, "A functional equation and its application to resource allocation and sequencing problems," *Management Sci.*, 16 (1969) 77-84.

[4] E. L. Lawler, "Sequencing to minimize the weighted number of tardy jobs," *RAIRO Rech. Oper.*, 10.5 Suppl. (1976) 27-33.

[5] E. L. Lawler, "Preemptive scheduling of uniform parallel machines to minimize the number of late jobs," Report 105/79, Mathematisch Centrum, Amsterdam, May 1979.

[6] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, "Recent developments in deterministic sequencing and scheduling: a survey," in *Deterministic and Stochastic Scheduling*, M. A. H. Dempster et al. (eds.), D. Reidel Publ. Co., 1982, pp. 35-73.

[7] E. L. Lawler, "Recent results in machine scheduling theory," *Proc. XI International Symposium on Mathematical Programming*, to appear.

[8] J. M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management Sci.*, 15 (1968) 102-109.

[9] J. B. Sidney, "An extension of Moore's due date algorithm," in *Theory of Scheduling and its Applications*, S. E. Elmaghraby (ed.), Lecture Notes in Economics and Mathematical Systems 86, Springer, Berlin, 1973, pp. 393-398.