Discrete Applied Mathematics 28 (1990) 45–57
North-Holland

MINIMIZING THE NUMBER OF TARDY JOB UNITS UNDER RELEASE TIME CONSTRAINTS

Dorit S. HOCHBAUM*
School of Business Administration and IEOR Department, University of California, Berkeley, CA 94720, USA

Ron SHAMIR**
School of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel

Received 27 October 1989

We study two single-machine scheduling problems: Minimizing the weighted and unweighted number of tardy units, when release times are present. Fast strongly polynomial algorithms are given for both problems: For problems with n jobs, we give algorithms which require $O(n \log n)$ and $O(n^2)$ steps, for the unweighted and weighted problems respectively. Our results also imply an extension of the family of very efficiently solvable transportation problems, as well as these which are greedily solvable using the "Monge sequence" idea.

1. Introduction

This paper studies two problems which arise in scheduling theory: The first is minimizing the number of tardy time units on a single machine, subject to release times. The second is the weighted version of the same problem. Preemption is allowed, and it is assumed that the total length of the jobs, N, is much greater than the number of jobs, n. In that situation, previously known algorithms, which consider each unit as a different job, become only pseudopolynomial and hence may be prohibitively inefficient. We provide ad-hoc algorithms which are strongly polynomial for both problems. Specifically, the unweighted problem is shown to be solvable in $O(n \log n)$ steps, and the weighted problem in $O(n^2)$ steps. The algorithm for the unweighted problem is also shown to be best possible in a common computational model. Our algorithms employ ideas from transportation and matching theory, as well as the UNION-FIND data structure for efficient set manipulation.

Our interest in the problems studied here is threefold:

– We identify new "high multiplicity" type problems which are solvable in strongly polynomial time.

* Supported in part by the Office of Naval Research under grant N00014-88-K-0377 and by the National Science Foundation under grant ECS-85-01988.
** Supported in part by Allon Fellowship.

- We extend the notion of greedily solvable transportation problems beyond the limits of previous studies, to include certain transportation problems in which some edges are forbidden.
- We identify new specially-structured transportation problems, which can be solved very fast by ad-hoc techniques.

In the remainder of this introduction, we shall elaborate on the issues mentioned above.

Assume that N unit-time jobs are to be scheduled on a single machine subject to release time constraints, so as to minimize the (weighted) number of tardy units. This problem is solvable by setting up an assignment problem whereby each job can be assigned to any allowable time unit with the appropriate weight, and the solution requires $O(N^3)$ steps. Assume now that there are only $n \ll N$ distinct *types* of unit-time jobs, where all units of the same type have identical parameters. This is the high multiplicity version of the above problem. The number of units of each type is called its multiplicity. Obviously the above solution as an assignment problem is applicable, again requiring $O(N^3)$ steps. However, the input now can be represented in length $O(n)$ numbers, hence that algorithm is not polynomial! Can we still solve the probem in a number of steps which will be polynomial in n and independent of N?

Questions of the high multiplicity (HM) type have been raised for many combinatorial optimization problems related to graph theory and scheduling (see [5] and the references thereof). In this paper we first address HM scheduling problems which involve release times, and show that strongly polynomial algorithms are obtainable for them too.

Note that the HM problems discussed here have another equivalent interpretation: According to the above definitions, there are n types of jobs, with p_i unit-time jobs from type i. The same problem can be represented as a problem with n jobs, were job i has length p_i, preemption is allowed (i.e., units of the same job do not have to appear contiguously in the schedule), and the cost criterion counts the number of tardy units. We shall use this formulation here. For more on the relations of the two interpretations and on applications of the HM problems, see [5].

As we shall show later, the problems posed here can be reformulated as transportation problems. This immediately allows the use of a strongly polynomial algorithm for the problem, e.g. [10] or [8]. The algorithms which we shall present here are at least an order of magnitude faster than the best strongly polynomial algorithms for the transportation problem. In that, the family of very efficiently solvable transportation problems is extended.

Our solution methods for the unweighted problem is based on the notion of greedily solvable transportation problems. A transportation problem is said to be greedily solvable by a permutation S of the decision variables if maximizing each of the decision variables in turn, according to the order prescibed by S, gives an optimum solution for every supply and demand vectors. Hoffman [6] gave a necessary and sufficient condition for a permutation S to provide an optimal solution. Such a permutation is called a Monge sequence. [1] gave an efficient polynomial algorithm which detects and constructs a Monge sequence if such exists. However, both studies assume that all entries in the transportation cost matrix are finite. In other words, there are no "forbidden edges" between supply and demand points, along which no shipping is possible. Hoffman [6] posed the question whether the idea of a Monge sequence (and perhaps the algorithm for constructing it) can be extended to problems with forbidden edges. For the unweighted scheduling problem studied here, we show that a Monge sequence does exist in a corresponding transportation problem, even though there are forbidden edges. Hence this extends the family of greedily solvable transportation problems, and is a first step towards answering Hoffman's question.[1]

2. The unweighted problem

In the scheduling problem of minimizing the number of tardy units, there are n jobs. Job j has release time r_j, due date d_j, and length p_j. All the numbers are assumed to be nonnegative integers. A *schedule* is an assignment of all the job units to distinct nonnegative integer time units. I.e., for $j=1,\dots,n$, job j is assigned to exactly p_j time units. (Note that the units of a job need not be contiguous in a schedule.) If a unit is assigned to time unit $[k,k+1)$, then we shall say it is assigned to time (or integer) k. We say that time t is *admissible* for job j if $r_j \le t$. A *feasible* schedule is a schedule in which all units are assigned to admissible times, i.e., for $j=1,\dots,n$, no unit of job j is assigned to time $k<r_j$. The *cost* of the schedule is the total number of units of jobs which are *tardy*, i.e., assigned to times greater than or equal to their respective due dates. The goal is finding a feasible schedule of minimum cost.

The same problem without release times has been observed to be solvable in $O(n \log n)$ steps by an algorithm which is a slight variation of the "earliest due date" rule [5].

Because of the release times constraints, it may happen that the machine will have idle periods in the schedules. That is, all job units may not appear contiguously in any optimal (or feasible) schedule. Moreover, the termination time of the optimal schedule is not immediately available. We shall first show that one can limit the search for an optimal solution to schedules which saturate a certain set of time intervals and idle the rest. These intervals can be determined by the following simple procedure: Schedule the jobs in increasing order of release times, contiguously from the first available and admissible time unit. The resulting filled intervals are the required ones. The procedure is described formally below:

[1] See concluding remark.

```
    Procedure INTERVALS
1     Reorder jobs so that $r_1 \le r_2 \le \cdots \le r_n$.
2     Set $T_1 \leftarrow r_1 + p_1$; $F_1 \leftarrow [r_1, T_1)$
3     For $i = 2, \dots, n$ do:
        If $T_{i-1} < r_i$ then set $R_{i-1} \leftarrow [T_{i-1}, r_i)$; $T_i \leftarrow r_i + p_i$; $F_i \leftarrow [r_i, T_i)$
        else set $R_{i-1} \leftarrow \emptyset$; $T_i \leftarrow T_{i-1} + p_i$; $F_i \leftarrow [T_{i-1}, T_i)$
      repeat
4     set $R_n \leftarrow [T_n, \infty)$.
```

The nonempty R- and F-intervals form a partition of $[r_1, \infty)$. We now show that in the search for an optimal schedule, one can ignore the R-intervals:

Lemma 2.1. *There exists an optimal schedule in which all the F-intervals are saturated (and thus all the R-intervals are idle).*

Proof. Let S be an optimal schedule. We shall show how to modify it so that it will satisfy the requirements of the lemma, without increasing its cost: Let $[t, t+1)$ be the first time unit in $\bigcup F_k$ which is idle in the schedule S. Then there exists a unit of some job j such that $r_j \le t$, which is scheduled to time $t' > t$. (This follows from the construction of the F-intervals.) That unit can be rescheduled earlier at time t, without destroying feasibility, and without increasing the cost of the schedule. By repeating this step we eventually obtain a schedule which saturates all the F-intervals. □

So, we can always preprocess the problem and find the F-intervals and the termination time. Once those are known, we can contract the R-intervals and obtain an equivalent problem for which there exists an optimal schedule with no idle time and with known termination time. Henceforth, we shall assume without loss of generality that the problem is already given in that form. The preprocessing requires linear time if the release times are sorted (they need to be sorted also for the optimization algorithms below).

Let N be the earliest completion time of a feasible schedule (as determined, for example, by the above procedure). We assume without loss of generality that the earliest time is zero, and that the latest due date is smaller than N. The set of integers $\tilde{S} = \{r_1, \dots, r_n, d_1, \dots, d_n, N\}$ includes all the time points at which the status of any job changes (from nontardy to tardy, or from inadmissible to admissible). After sorting $\tilde{S}$ and omitting identical numbers, one gets the order: $0 = u_0 < u_1 < u_2 < \cdots < u_m = N$. This forms a partition of the interval $[0, N)$ into subintervals $I_i = [u_{i-1}, u_i)$, $i = 1, \dots, m$. We shall call these the *intervals of the problem*. Within each of these intervals, the status of all the jobs does not change.

Using these intervals, we can reformulate the problem as a transportation problem as follows: Let X_{ji} be the number of units of job j processed in interval i.

Denote by C_{ji} the contribution to the cost by a unit of job j scheduled in interval i. Namely:

$$C_{ji} = \begin{cases} 0, & \text{if } r_j < u_i \leq d_j, \\ 1, & \text{if } u_i > d_j. \end{cases}$$

Let $l_i = u_i - u_{i-1}$. The formulation is:

$$\begin{aligned} \text{minimize} \quad & \sum_{j,i} X_{ji} C_{ji}, \\ \text{subject to} \quad & \sum_i X_{ji} = p_j, \quad j = 1, \dots, n, \\ & \sum_j X_{ji} = l_i, \quad i = 1, \dots, m, \\ & X_{ji} \geq 0 \text{ and integer}, \ j = 1, \dots, n, \ i = 1, \dots, m. \end{aligned}$$

(In the second set of constraints, equality is guaranteed by the preprocessing and idle intervals contraction, as discussed following Lemma 2.1.)

The fastest strongly polynomial transportation algorithm can be used to solve this problem in $O(n^3 \log n + n^2 \log^2 n)$ operations [8]. (Orlin's algorithm applies to the more general minimum cost flow problem, but is also the fastest strongly polynomial algorithm for transportation problems.) The algorithm described below solves the same problem in $O(n \log n)$ operations.

The algorithm works by assigning groups of units from each job to these intervals, in a prescribed order. The algorithm handles the jobs in decreasing order of release times. It schedules all the units of the current job to intervals and then proceeds to the next job. For each job, the units are assigned to intervals in decreasing order of intervals from the due date backwards, and when no nontardy time slots are available from time N backwards. The algorithm is greedy in the sense that for each job and in each interval, it always assigns maximum number of units to the current interval. The algorithm outputs an $n \times m$ matrix S, where S_{ji} is the number of units of job j to be scheduled in interval i (i.e., $S_{ji} = X_{ji}$ in the transportation formulation). The schedule produced by the algorithm is completely determined by the matrix of values (S_{ji}). This follows since within each interval $[u_i, u_{i+1})$ no release time or due date appears. Hence the cost does not depend on the internal ordering of the units of the various jobs within each of the intervals.

```
    Algorithm A
1     Reorder jobs so that $r_1 \geq r_2 \geq \cdots \geq r_n$.
2     Sort the integers $\{r_1, \dots, r_n, d_1, \dots, d_n, N\}$. Let the sorted order (with
        identical numbers omitted) be $0 = u_0 < u_1 < u_2 < \cdots < u_m = N$.
      For $i = 1, \dots, m$, set $l_i = u_i - u_{i-1}$
3     For $j = 1, \dots, n$ do:
        Find $k$ satisfying $d_j = u_k$, and $q$ satisfying $r_j = u_q$.
        For $i = k, k-1, k-2, \dots, q+1, m, m-1, \dots, k+1$ do:
            $\Delta \leftarrow \min\{l_i, p_j\}$
            $S_{ji} \leftarrow \Delta; l_i \leftarrow l_i - \Delta; p_j \leftarrow p_j - \Delta$
        **repeat**
    **repeat.**
```

Table 1 gives the scanning order of the algorithm in an example with four jobs. The order of the release times and due dates is: $r_4 < r_3 < r_2 < d_3 < r_1 = d_4 < d_2 < d_1$. (Note that the numerical values are not necessary in order to determine the scanning order.) Example 2 (see Table 2) gives the optimal solution of that problem for specific lengths of jobs and intervals lengths. Here $p = (p_1, p_2, p_3, p_4) = (6, 1, 7, 7)$, $r = (12, 7, 4, 0)$ and $d = (17, 15, 10, 12)$.

Let us now prove the validity of the algorithm. The proof will follow the lines of [5] for the weighted problem without release times, with appropriate extensions.

Theorem 2.2. *Algorithm* A *provides an optimal solution.*

Proof. Assume there exists a schedule (Y_{ji}) with cost strictly lower than that of the algorithmic solution S. Algorithm A determines a unique scanning order of the pairs of job and interval (j, i). Compare the two schedules according to that order, and let (α, β) be the first pair on which the two solutions differ. Then $S_{\alpha\beta} > Y_{\alpha\beta}$, since Algorithm A assigns the maximum possible number of units to each variable in turn. Since $\sum_{i=1}^{m} S_{\alpha i} = \sum_{i=1}^{m} Y_{\alpha i} = p_\alpha$, there must be a pair (α, γ) which is scanned later in the scanning order, for which $Y_{\alpha\gamma} > 0$. By a similar argument, since by Lemma 2.1 the column totals in the two schedules must also be equal, there must be a pair (δ, β) for which $Y_{\delta\beta} > 0$ and (δ, β) is also scanned after (α, β).

Now, make the following augmentation in the Y solution:

$$Y_{\alpha\beta} \leftarrow Y_{\alpha\beta} + 1,$$
$$Y_{\alpha\gamma} \leftarrow Y_{\alpha\gamma} - 1,$$
$$Y_{\delta\beta} \leftarrow Y_{\delta\beta} - 1,$$
$$Y_{\delta\gamma} \leftarrow Y_{\delta\gamma} + 1.$$

Table 1. Example 1: Scanning order of Algorithm A. Numbers inside the table indicate the place of that (job, interval) pair in the scanning order.

Interval	1	2	3	4	5	6	7
Interval endpoints	u_0	u_1	u_2	u_3	u_4	u_5	u_6
Interval endpoints	r_4	r_3	r_2	d_3	$r_1 = d_4$	d_2	d_1
job 1					2	1	3
job 2			6	5	4	8	7
job 3		10	9	14	13	12	11
job 4	18	17	16	15	21	20	19

Table 2. Example 2: Solution generated by Algorithm A. Ordering of release times and due dates is the same as in Example 1. Number in location (j,i) is the number of units of job scheduled to interval i.

Interval		1	2	3	4	5	6	7
Length		4	3	3	2	3	2	4
Job no.	Length							
1	6					3	2	1
2	1				1			
3	7		3	3				1
4	7	4			1			2

The subtractions are possible since $Y_{\alpha\gamma}>0$ and $Y_{\delta\beta}>0$. The last addition is possible by the ordering of the jobs according to decreasing release dates: It guarantees that in interval γ, job δ is admissible (that is, $u_{\gamma-1}\geq r_\alpha \geq r_\delta$), and so $Y_{\delta\gamma}$ may attain a positive value without destroying the feasibility of the schedule.

The change in the solution cost due to the augmentation is then

$$C_{\alpha\beta}-C_{\alpha\gamma}+C_{\delta\gamma}-C_{\delta\beta}.$$

Case 1: $\gamma<\beta$. In that case $C_{\alpha\beta}=C_{\alpha\gamma}$, since the scanning order requires scanning intervals of equal costs in decreasing order. Hence $C_{\delta\gamma}\leq C_{\delta\beta}$, so the cost may only decrease.

Case 2: $\gamma>\beta$. In that case $C_{\alpha\beta}=0$, $C_{\alpha\gamma}=1$. Also $C_{\delta\gamma}-C_{\delta\beta}\leq 1$, so again the cost may only decrease.

By repeating the above procedure, either we get a solution of lower value, or we eventually obtain a solution with the same cost as Y which has a cost identical to that of S, and in both cases we get a contradiction. $\square$

Note that unlike the results in [5], the above results cannot be directly obtained from Hoffman's theory on Monge sequences [7]. This is because the costs C_{ji} are not defined for all i,j: If interval i precedes the release time of job j, no units of the job can be assigned to that interval. There are some intervals for which certain jobs cannot be assigned (in other words, $C_{ji}=\infty$ for $r_j\geq u_i$). Hoffman's theorem on the existence of a Monge sequence requires that all costs be finite. The same requirement is necessary for the algorithm described in [1] for detecting such a sequence. However, the above algorithm does provide a "Monge sequence" of all finite (nonforbidden) (i,j). This is possible only because of the specific ordering of the release dates, which guarantees that augmentation is indeed possible.

Let us now turn to the complexity of the algorithm. Ordering the release times and due dates (steps 1 and 2) requires $O(n\log n)$ operations. A table which contains the number of each release time and due date in the order u is constructible in $O(n)$, and it facilitates finding k and q is step 3 in constant time. So in step 3, for each pair of job and interval, the work required is at most a constant. Hence this step requires a total of $O(n^2)$ operations in a straightforward implementation. We will show how to implement step 3 in a linear number of operations. Observe first that whenever a value of some S_{ji} is increased, either all the units of a job have been scheduled or an interval has been saturated (or both). Hence at most $n+m-1=O(n)$ values of S_{ji}'s will be positive in the solution. We would like to avoid the necessity to scan all those pairs (j,i) for which $S_{ji}=0$. We shall use the UNION-FIND algorithm [10] for this purpose.

Recall that the UNION-FIND algorithm handles elements which are partitioned into sets. The algorithm supports two operations: FIND(i), finding the set to which element i belongs, and UNION(S_1, S_2, S_3), replacing the disjoint sets S_1 and S_2 by the set $S_3 = S_1 \cup S_2$. In our case, *elements* will correspond to intervals, and each *set* will correspond to a (contiguous) set of intervals. The first of these intervals is nonsaturated, where all the ones that follow, if there are any, are saturated and hence no longer available. In addition, each set will have a label. The *label* of a set will be the number of the first (i.e., smallest numbered) interval in that set, which is the only one available in that set. The partition into sets will maintain the following invariant property: For all the intervals in each set, the label of that set gives the largest numbered free (i.e., nonsaturated) interval which is not later than each of the intervals. We shall assume that operation FIND(i) retrieves the *label* of the set to which interval i belongs. So performing the operation FIND(i) immediately gives the number t of the first free interval prior to i (including i), thereby avoiding the need to scan the nonfree intervals $i, i-1, i-2, \ldots, t+1$. (If interval 1 is saturated, then FIND(1) gives FIND(m), the rightmost interval.) When interval t has been saturated and becomes nonfree, if v is the set which includes interval $t-1$, the operation UNION(t, v, v) form the union of the set which includes t with the set which includes $t-1$, and gives to the new set the label of the latter. The revised algorithm is presented formally below.

```
        **Algorithm A** (revised)
1         Reorder jobs so that $r_1 \geq r_2 \geq \cdots \geq r_n$.
2         Sort $\{r_1, \ldots, r_n, d_1, \ldots, d_n, N\}$ to get $0 = u_0 < u_2 < \cdots < u_m = N$.
          **For** $i = 1, \ldots, m$, set $l_i \leftarrow u_i - u_{i-1}$
3         **For** $i = 1, \ldots, m$ **do**: Form a set labeled $i$ which contains $\{i\}$.
4         **For** $j = 1, \ldots, n$ **do**:
4.1         Find $k$ satisfying $d_j = u_k$, and $q$ satisfying $r_j = u_q$.
4.2         $t \leftarrow$ FIND($k$). **If** $t < q$ **then** $t \leftarrow$ FIND($m$)
4.3         **while** $p_j > 0$ **do**:
4.4           $\Delta \leftarrow \min\{l_t, p_j\}$
4.5           $S_{jt} \leftarrow \Delta$; $l_t \leftarrow l_t - \Delta$; $p_j \leftarrow p_j - \Delta$
4.6           **if** $l_t = 0$ **then** $v \leftarrow$ FIND($t-1$);
              **if** $v < q$ **then** $v \leftarrow$ FIND($m$); UNION($t, v, v$)
4.7           $t \leftarrow v$
            **repeat**
          **repeat.**
```

Table 3. Example 3: Partition into sets by the revised algorithm. For each state and interval i, the number in the table is FIND(i).

Interval	1	2	3	4	5	6	7
Initial state	1	2	3	4	5	6	7
State after scheduling job 1	1	2	3		4		7
State after scheduling job 2	1	2	3		4		7
State after scheduling job 3		1			4		7

Example 3 (see Table 3) demonstrates the evolvement of the set partition during the performance of the revised algorithm, for the problem described in Example 2.

The algorithm performs at most $m-1$ UNION operations and $2n+2m$ FIND operations. The union operation is performed on sets of adjacent intervals. As such the union tree is a path and the UNION-FIND algorithm performs all these steps in $O(n)$ operations [4]. So, step 4 of the algorithm is implementable in $O(n)$ steps, and we conclude:

Theorem 2.3. *Algorithm* A *can be implemented in* $O(n \log n)$ *operations.*

Note that the preliminary work of sorting the r and d vectors is done only once, and is independent of the job lengths. If one has to solve the same problem several times with different job lengths, the work per each subsequent solution after the first one will be $O(n)$ only.

Note also that if the unweighted problem could be solved faster than in $O(n \log n)$ steps, that solution could be used to sort n numbers in the same time complexity: Given n numbers $\alpha_1, \dots, \alpha_n$, form a scheduling problem of minimizing the number of tardy units with n jobs where $p_i = 1$, $r_i = n\alpha_i$ and $d_i = (n+1)\alpha_i$ for $i = 1, \dots, n$. The optimal solution to that problem has value zero, and the order of the jobs in an optimal solution (ignoring empty subintervals) corresponds to a sorting of the numbers $\alpha_1, \dots, \alpha_n$. Hence $O(n \log n)$ is also a lower bound for the complexity of our problem, at least under the computational model of comparisons.

3. The weighted problem

We now address the scheduling problem of minimizing the weighted number of tardy units in the presence of release times. Let us first fix notation: The input data includes n jobs, where job j has release time r_j, length p_j, due date d_j and weight w_j. All numbers are nonnegative integers. A feasible schedule in defined as in Section 2. The cost of each tardy unit of job j will now be w_j (instead of 1 in Section 2), and we look for a feasible schedule of minimum total cost.

The weighted problem without release times has been shown to be solvable in $O(n \log n)$ time by an ad-hoc algorithm [5], which may be interpreted as constructing a Monge sequence [6] for a corresponding transportation problem. As we shall see, the same technique is not applicable here.

This problem can also be recast as a transportation problem. The formulation is identical to the one in Section 2, only this time the costs are weighted:

$$C_{ji} = \begin{cases} 0, & r_j < u_i \le d_j, \\ w_j, & d_j < u_i. \end{cases}$$

Using the fastest strongly polynomial transportation algorithm [8], a solution can again be obtained in $O(n^3 \log n + n^2 \log^2 n)$ steps. We shall give an $O(n^2)$ algorithm for this problem. The faster algorithm of the previous section is not applicable here, because of the weights. First let us observe a special case in which Algorithm A does solve the weighted problem:

Proposition 3.1. *If for all i,j, $w_i \ge w_j$ whenever $r_i \ge r_j$, Algorithm* A *provides an optimal solution in* $O(n \log n)$ *operations.*

Proof. The same reasoning as in the proof of Theorem 2.2 follows, with the more general cost matrix. The order of weights guarantees that the same augmentation in the Y solution will still result in an equal or lower cost. More precisely, if in Case 2 of that proof $C_{\alpha\beta} = 0$ and $C_{\alpha\gamma} = w_\alpha$, then $C_{\delta\gamma} - C_{\delta\beta} \le w_\delta$. Since $w_\delta \le w_\alpha$, the result follows. □

The algorithm for the general problem uses Algorithm A as a subroutine. It proceeds in phases, as follows: The jobs are assumed to be numbered in decreasing order of weights. After phase k, the number of nontardy units in an optimal solution has already been determined for each of the first k jobs. Let these numbers be $\pi_1, \dots, \pi_k$. In phase $k+1$, job $k+1$ is introduced and Algorithm A is used to solve the *unweighted* problem on jobs $1, 2, \dots, k+1$, where job $k+1$ has p_{k+1} units and job i has π_i units for $i = 1, \dots, k$. The solution to that unweighted subproblem determines π_{k+1}, the maximum number of nontardy units of job $k+1$ (given that the numbers of nontardy units for each of the previous k jobs are fixed) and the algorithm proceeds to the next phase.

```
        Algorithm B
1         Reorder jobs so that $w_1 \ge w_2 \ge \cdots \ge w_n$.
2         Set $\pi_1 \leftarrow \min\{p_1, d_1 - r_1\}$, cost = $w_1(p_1 - \pi_1)$
3         For $k = 2, \dots, n$ do:
            Use Algorithm A to solve the unweighted subproblem on jobs
              $1, \dots, k$ with lengths $\pi_1, \dots, \pi_{k-1}, p_k$. Let $t$ be the number of tar-
              dy units in the solution.
            Set $\pi_j \leftarrow p_j - t$, cost $\leftarrow$ cost + $t \times w_j$
          repeat.
```

In order to prove the validity of the algorithm, we shall reformulate the problem as a matching problem: Let $G=(V, W, E)$ be a bipartite graph, where V and W denote the two parts of its vertex set and E is the edge set. Denote by $V' \subset V$ a proper subset of V, and denote its complement $V-V'$ by V_1. $G'=(V', W, E')$ is the subgraph induced by the vertices $V' \cup W$, i.e. by removing V_1 from V. In our case the graph G describes the problem in phase $k+1$: the vertices in V correspond to all the units of jobs $1, \dots, k+1$, and the vertices in W correspond to all the time units $0, 1, \dots, N-1$. A vertex of job unit j is connected by an edge to the vertex of time unit t if at that time unit, job j is admissible and nontardy (i.e., $r_j \le t < d_j$). V_1 corresponds to all the units of job $k+1$. Hence G' is the graph which describes the problem in phase k. A maximum matching in G corresponds to a schedule in which the number of nontardy job units is maximized, or, equivalently, a schedule which minimizes the number of tardy units. We would like to show that a maximum matching M' on G' can be extended to a maximum matching on G, without exposing any vertex of V' which was matched in M'.

Lemma 3.2. *Using the above notation, if M' is a maximum matching on G', then there exists a maximum matching M on G such that for every $v \in V'$, if v is matched in M', then it is also matched in M.*

Proof. Let M be any maximum matching on G. We shall show how to modify it such that it will satisfy the theorem. Let $\hat{M}$ be the symmetric difference of the two matchings M and M'. Each connected component of the subgraph induced by $\hat{M}$ is either (1) an isolated vertex or (2) an even elementary cycle whose edges are alternately in M and M', or (3) an elementary alternating path whose endpoints are distinct and are both unmatched in one of the two matchings (see, e.g. [2, p. 123]).

Vertices of V' which were matched in M' and appear in type (1) or (2) components are also matched in M as required. The only ones which may be unmatched in M are *endpoints* of type (3) components, an alternating path. Denote such a path by $(v_0, e_1, v_1, e_2, v_2, \dots, e_j, v_j)$, and assume that $v_0 \in V'$ is unmatched in M. The path length, j, is even or else we get a contradiction to the maximality of M. Hence by setting

$$M \leftarrow (M - \{e_2, e_4, \dots, e_j\}) \cup \{e_1, e_3, \dots, e_{j-1}\}$$

we get a matching with the same cardinality as the original M, which matches all the vertices in that path which are in V' and were matched in M'. (Note that v_j was not matched in M'.)

By repeating the above process we eventually obtain the required maximum matching M, which matches all the vertices in V' that were matched in M'. □

We can now prove the validity of the theorem:

Theorem 3.3. *Algorithm* B *provides an optimal solution to the weighted problem.*

Proof. The proof is by induction on the number k of phases: Denote by P_k the weighted subproblem with jobs $1, \ldots, k$ only. For $k=1$, step 2 of the algorithm clearly gives an optimal solution to P_1. Assume that after phase k of the algorithm, the numbers of nontardy units from jobs $1, \ldots, k$ in an optimal solution to P_k are $\pi_1, \ldots, \pi_k$ respectively. Algorithm A is used in phase $k+1$ and a maximum matching is found. By Lemma 3.2 there exists such maximum matching in which $\pi_1, \ldots, \pi_k$ units of jobs $1, \ldots, k$ are still matched. Since the weight of job $k+1$ is the smallest among jobs $1, \ldots, k+1$, this gives an optimal solution to the weighted scheduling problem P_{k+1}. □

Let us now turn to the complexity of the algorithm: Steps 1 and 2 of the algorithm require $O(n \log n)$. Step 3 requires solutions of $n-1$ unweighted problems of the type analyzed in Section 2. Since the release times and due dates of all these $n-1$ subproblems are all subsets of the *same* single set $\{r_1, \ldots, r_n, d_1, \ldots, d_n\}$, that set needs to be ordered only once, and the remaining work per each solution is $O(n)$, by the proof of Theorem 2.2. Hence the total work in step 3 is $O(n^2)$, which is also the overall complexity. Hence we conclude:

Theorem 3.4. *Algorithm* B *solves the weighted problem in* $O(n^2)$ *operations.*

In order to find a schedule (S_{ji}) during the execution of the algorithm, note that whenever a number of units of job j are determined to be tardy, this implies that the interval $[r_j, d_j)$ has been completely filled. Hence the schedule in it has been completely found by Algorithm A.

Remark added in the revision. After the completion of this work, the theory of Monge sequences has been extended to transportation problems with forbidden arcs [3, 9]. For the unweighted case discussed in Section 2, the scanning order of the variables in our solution indeed turns out to be a Monge sequence in that extended sense. For the weighted case of Section 3, one can easily demonstrate that for some problems no single optimal scanning order exists. This follows since a necessary condition for the existence of such a scanning order in a problem with forbidden arcs is violated. Hence it is impossible to find a faster algorithm for the weighted problem along the lines of our solution to the unweighted one.

Acknowledgement

We would like to thank Amos Fiat for bringing to our attention reference [4].

References

[1] N. Alon, S. Cosares, D. Hochbaum and R. Shamir, An algorithm for the detection and construction of Monge sequences, Linear Algebra Appl. 114/115 (1989) 669–680.

[2] C. Berge, Graphs and Hypergraphs (North-Holland, Amsterdam, 1973).
[3] B.L. Dietrich, Monge sequences, antimatroids, and the transportation problem with forbidden arcs, Rept., IBM T.J. Watson Research Center (1989).
[4] H.N. Gabow and R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. System Sci. 30 (1985) 209-221.
[5] D.S. Hochbaum and R. Shamir, Strongly polynomial algorithms for the high multiplicity scheduling problem, Rept. TR-100/88, Computer Science Institute, Tel Aviv University, Tel Aviv (1988); also: Oper. Res., to appear.
[6] A. Hoffman, On simple transportation problems, in: V. Klee, ed., Convexity, Proceedings of Symposia in Pure Mathematics 7 (Amer. Math. Soc., Providence, RI, 1963) 317-327.
[7] A. Hoffman, Private communication.
[8] J.B. Orlin, A faster strongly polynomial miminum cost flow algorithm, in: Proceedings 20th ACM Symposium of the Theory of Computing (1988) 377-387.
[9] R. Shamir, A fast algorithm for constructing Monge sequences in transportation problems with forbidden arcs, Rept. 136/89, Institute of Computer Science, Tel Aviv University, Tel Aviv (1989); also: Discrete Math., to appear.
[10] E. Tardos, A strongly polynomial minimum cost circulation algorithm, Combinatorica 5 (1985) 247-255.