Mathematical Programming 11 (1976) 229–251.
North-Holland Publishing Company

A DUAL ALGORITHM FOR THE ONE-MACHINE SCHEDULING PROBLEM*

Marshall L. FISHER
University of Pennsylvania, Philadelphia, Pa., U.S.A.

Received 20 January 1974
Revised manuscript received 18 March 1976

A branch and bound algorithm is presented for the problem of scheduling n jobs on a single machine to minimize tardiness. The algorithm uses a dual problem to obtain a good feasible solution and an extremely sharp lower bound on the optimal objective value. To derive the dual problem we regard the single machine as imposing a constraint for each time period. A dual variable is associated with each of these constraints and used to form a Lagrangian problem in which the dualized constraints appear in the objective function. A lower bound is obtained by solving the Lagrangian problem with fixed multiplier values. The major theoretical result of the paper is an algorithm which solves the Lagrangian problem in a number of steps proportional to the product of n^2 and the average job processing time. The search for multiplier values which maximize the lower bound leads to the formulation and optimization of the dual problem. The bounds obtained are so sharp that very little enumeration or computer time is required to solve even large problems. Computational experience with 20-, 30-, and 50-job problems is presented.

1. Introduction

The one-machine scheduling problem requires a specification of the order in which n jobs are to be processed on a single machine. The processing time and due date of each job are known integers, and a job's tardiness is the amount by which its completion time exceeds its due date. An optimal solution is a sequence for performing the jobs in which the sum of the tardiness over all jobs is minimum. This difficult combinatorial problem has a long history. Algorithms have been proposed by Elmaghraby [4], Emmons [5], Held and Karp [14], Lawler [19], and Srinivasan [24]. Baker and Martin [1] report computational experience using some of these algorithms on problems of up to 15 jobs.

We will describe an algorithm for the one-machine scheduling problem which has performed remarkably well on a sample of large and difficult problems. The heart of the algorithm is a method for obtaining an extremely sharp lower bound on the optimal objective value. This lower bound is used for fathoming in a branch and bound procedure. The bound is obtained using a dual problem of the type described in Fisher [6] for the more general resource-constrained network scheduling problem. In this dual problem we regard the machine on which jobs are to be processed as imposing a constraint for each time interval $[t, t+1]$ for

* This work was supported in part by National Science Foundation Grant SOC-7402516.

$t = 0, 1, 2, \ldots$. A dual variable is associated with each of these constraints and used to form a Lagrangian problem in which the dualized constraints appear in the objective function. In reference [5] Emmons has given results for determining a partial ordering of the jobs which is satisfied by at least one optimal solution. The feasible region of the Lagrangian problem is the set of non-negative integer starting times for the n jobs which satisfy the Emmons partial ordering. Solution of the Lagrangian problem with any fixed values for the dual variables produces a lower bound on the optimal objective value. The dual problem is to find values for the dual variables for which the lower bound is maximum. Our objective in this derivation is to obtain a Lagrangian problem which contains much of the information in the constraints of the original problem and yet is easy to solve. The major theoretical result of this paper is an algorithm which solves the Lagrangian problem in a number of steps proportional to the product of n^2 and the average job processing time. As a practical matter, the magnitude of the average processing time can always be controlled if one is willing to accept a sufficiently coarse unit of measure for process times. Good or optimal solutions to the dual problem are obtained using a subgradient method of Held, Wolfe and Crowder [16]. This method uses the Lagrangian algorithm as a subroutine.

Because of the underlying nonconvexity of the one-machine scheduling problem, the optimal value of the dual problem may not equal the optimal value of the one-machine problem and we must resort to branch and bound to obtain the optimal solution. However, the dual bounds are sufficiently sharp that the optimal solution is usually obtained with little, if any, branching. We will use a particular 50 job problem as an illustrative example throughout the paper. (This problem is described in the third line from the bottom of Table 3 in Section 5.) Our computational experience with this problem is typical. The optimal value is 1464 and the lower bound obtained from the dual problem is 1459. Even though this is one of the more difficult of the problems we solved (only 5 of 75 problems had longer solution times), the enumeration tree contains only 30 nodes and is given in Section 4 in its entirety.

Generally, we have focused on large problems in our computational work. A recent text on sequencing theory ([2], p. 65] suggests that the solution of problems with more than 20 jobs might be prohibitively expensive with existing algorithms. The algorithm given here has been applied to a sample of 75 problems equally distributed over 3 problem sizes, $n = 20$, 30, and 50. Optimal solutions were obtained for all problems on a 360-67 in average times of 2.82, 10.08, and 63.49 seconds respectively for the three problem sizes. The lower bound obtained from the dual problem was on average equal to 99.6% of the optimal value. Additionally, a feasible solution is generated each time a Lagrangian problem is solved by sequencing jobs in order of their starting times in the Lagrangian solution. The value of the best feasible solution generated before any branching was on average equal to 100.2% of the optimal value. Using the algorithm in this way to generate feasible solutions would appear to allow the practical solution of very large problems since a comparitively small amount of time is required to obtain these solutions.

The general approach which we use is applicable to many combinatorial

problems. General discussions of the use of dual problems in discrete optimization are given in [7] and [11]. Specific applications include the traveling salesman problem [15], the general resource-constrained network scheduling problem [6] and the general integer programming problem [8]. The work of Held and Karp was initiative in this area. Our approach for obtaining lower bounds could also be used for the expanded scheduling problem treated by Gelders and Kleindorfer [10]. One purpose of this paper is to demonstrate the value of this dual approach for other combinatorial problems, particularly for the resource-constrained network scheduling problem.

Shwimer [23] and Rinnooy Kan et al. [21] have given algorithms for a more general problem in which an importance weight w_j is associated with each job and we are required to minimize the weighted sum of the tardiness over all jobs. The algorithm given here is applicable to this problem with minor changes. These changes will be noted at the appropriate places in the paper.

Section 2 gives results from [5] for ordering jobs and discusses their use in this paper. In Section 3 the Lagrangian and dual problems are derived and algorithms for their solution presented. Section 4 gives the branch and bound algorithm. Section 5 contains a discussion of our computational experience.

2. Reduction and sequencing

Common sense would lead us to certain suppositions about the order of jobs in an optimal solution. For example, we would expect a job with a very small due date to precede one with a very large due date. In [5] Emmons pursues this type of reasoning formally and extensively to develop conditions under which one job precedes another in some optimal solution. Let p_j and d_j denote the given process time and due date of job j, and let $T = \Sigma_{j=1}^{n} p_j$. Let β_j denote a set of jobs which are required to precede job j, α_j the corresponding set $\{i\colon j \in \beta_i\}$ of jobs which are required to follow job j and α_j' the complement of α_j with respect to $\{1, \ldots, n\}$. These sets are initially empty and are augmented by successive application of Theorem 2.1.

Theorem 2.1. *Job j precedes job k in some optimal solution if any of the conditions* (2.1)–(2.3) *are satisfied.*

$$p_j \leq p_k \quad \text{and} \quad d_j \leq \max\left(p_k + \sum_{i \in \beta_k} p_i,\, d_k\right). \tag{2.1}$$

$$p_j > p_k, \qquad d_j \leq d_k, \qquad d_k + p_k \geq \sum_{i \in \alpha_j'} p_i. \tag{2.2}$$

$$d_k \geq \sum_{i \in \alpha_j'} p_i. \tag{2.3}$$

This theorem contains the results of Emmons with some minor extensions. Theorem 2.1 is also valid for the weighted tardiness problem if we simply append the condition $w_j \geq w_k$ to (2.1) and (2.2). The theorem would then give the results

developed in [21]. The theorem can be proven by assuming the existence of an optimal solution in which job k precedes job j and showing that if (2.1), (2.2) or (2.3) are satisfied the order of j and k may be reversed without increasing tardiness. See reference [5] or [21] for details.

By reversing the roles of j and k in Theorem 2.1 we may give a second test for k to precede j in an optimal solution. In general three possibilities can occur. Precisely one of the two tests can be satisfied, both can be satisfied (the order of j and k is irrelevant), or neither can be satisfied. It is the last possibility which prevents us from using Theorem 2.1 to give a polynomially bounded algorithm for the minimum tardiness problem.

Theorem 2.1 is used here in two ways. First, following essentially the procedure suggested by Emmons, we try to reduce the size of the problem by finding a job which precedes or succeeds all other jobs in an optimal schedule. Such a job may be placed first or last and removed from the problem. When no further progress can be made with this reduction algorithm we use the following sequencing algorithm to create a partial ordering of the jobs. Henceforth we assume n and T have been modified to reflect any problem reduction which has taken place.

Sequencing algorithm

Step 1. Initialize $\beta_i = \emptyset$, $i = 1, \ldots, n$ and $J = \{1, \ldots, n\}$.

Step 2. Select j to be the index in J with minimum value of $\Sigma_{i \in \beta_j} p_i$. If there are ties, select the minimum index. Set $J = J \sim \{j\}$. (Here and elsewhere in the paper for general sets A and B we let $A \sim B$ denote the set $\{i \in A: i \notin B\}$.)

Step 3. For each $k \neq j$ for which $k \notin \beta_j$ and $j \notin \beta_k$, test the conditions of Theorem 2.1 for j to proceed k. If they are satisfied then set $\beta_i = \beta_i \cup \{j\} \cup \beta_j$ for $i \in \{k\} \cup \alpha_k$. If j has been placed in any β_k during this execution of step 3, repeat the step. Otherwise, go to step 4.

Step 4. If $J \neq \emptyset$, go to step 2. Otherwise go to step 5.

Step 5. If no elements have been added to any set β_i, $i = 1, \ldots, n$ since the last initialization of J, stop. Otherwise, reinitialize $J = \{1, \ldots, n\}$ and go to step 2.

The selection rule in step 2 is intended to enhance the effectiveness of Theorem 2.1 by increasing the likelihood that β_k will be large in (2.1) and α'_j small in (2.2) and (2.3).

The reduction and sequencing procedures will be illustrated with the example cited in Section 1. This example is described by the third line from the bottom of Table 3 in Section 5. The process times for the example are given by

(3, 4, 5, 4, 3, 4, 6, 1, 3, 5, 1, 3, 1, 1, 1, 2, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4,
5, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10)

and the due dates by

(67, 67, 69, 72, 73, 74, 74, 76, 78, 78, 80, 81, 99, 103, 108, 84, 107,
105, 86, 98, 103, 106, 107, 109, 112, 115, 101, 106, 107, 108, 80, 84, 115,
83, 92, 97, 103, 114, 76, 96, 100, 105, 114, 83, 93, 104, 105, 66, 76, 87).

When the reduction algorithm is applied to this problem the first 12 jobs are placed at the beginning of the solution in the order they appear above and the last job is placed at the end. Let $\bar{\beta}_i \subseteq \beta_i$ denote the set of *immediate* predecessors of job i. These are jobs $j \in \beta_i$ such that there exists no $l \in \beta_i$ with $j \in \beta_l$ also. Further $\bar{\alpha}_i$ will denote the immediate successors of i and is given by $\bar{\alpha}_i = \{j\colon i \in \bar{\beta}_j\}$. The sets $\bar{\beta}_i$ obtained by the sequencing algorithm for the 37 remaining jobs (reindexed from 1 to 37 in the order they appear above) are represented by the directed graph in Fig. 1. In this graph vertices correspond to jobs, job numbers are shown inside each vertex, and an edge exists from i to j if and only if $i \in \bar{\beta}_j$. For the moment, ignore any distinction between solid and dashed edges. To make sense such a graph must be acyclic. This property is guaranteed by the way in which the sets β_i are updated in step 3 of the sequencing algorithm.

3. Derivation and solution of the dual problem

The precedence sets β_i and α_i imply constraints which may be added to the one machine scheduling problem without loss of optimality. These constraints will make the problem more suitable for our purposes. Let x_i denote the start time of job i and $x = (x_1, \ldots, x_n)$. A lower bound on x_i is given by

$$a_i = \sum_{j \in \beta_i} p_j.$$

It is easy to show that in an optimal solution jobs are processed consecutively with no intervening idle time. This means that the last job is completed at time T

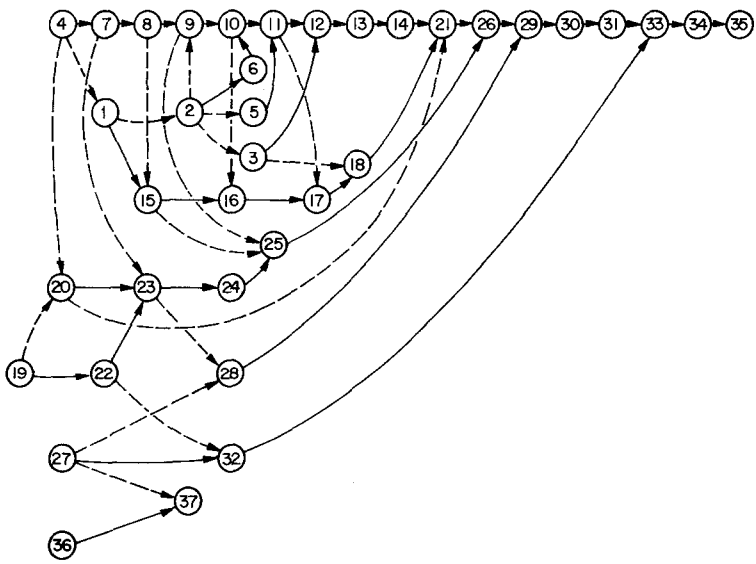

Fig. 1. Precedence graph for the example.

and an upper bound on x_i is given by

$$b_i = T - p_i - \sum_{j \in \alpha_i} p_j.$$

In terms of x, the sets $\bar{\alpha}_i$ imply constraints of the form $x_j \geq x_i + p_i$ for $j \in \bar{\alpha}_i$. If all sets $\bar{\alpha}_i$ satisfy $|\bar{\alpha}_i| \leq 1$ then a very efficient algorithm can be given for optimizing over the set of x which satisfy these constraints. To obtain sets with this property we simply delete a sufficient number of elements from each $\bar{\alpha}_i$. Let $A_i \subseteq \bar{\alpha}_i$, $i = 1, \ldots, n$ denote sets obtained by such a deletion so that $|A_i| \leq 1$. Let $B_i = \{j: i \in A_j\}$ denote the corresponding predecessor sets. For example, if the dashed edges in Fig. 1 are deleted, then the successor sets defined by the remaining graph have the required property. Of course there are many sets of edges which could be deleted to obtain the required property. In our computational work edges were removed by a rule which preserved longest paths in the network.

Let

$$X = \{x: a_i \leq x_i \leq b_i, x_j \geq x_i + p_i \text{ for } j \in A_i, x_i \text{ integer}, i = 1, \ldots, n\}.$$

While X expresses some of the feasibility requirements of the problem, in general an $x \in X$ will not be feasible. To obtain the additional constraints required for feasibility we define for $x \in X$ and integer $t \in [1, T]$ the set $S_t(x) = \{i: t - p_i \leq x_i \leq t - 1\}$ of jobs in process during the interval $[t-1, t]$ and let $g_t(x) = |S_t(x)|$, the number of jobs in process during this interval. The fact that jobs are processed continuously in an optimal schedule means that we can require $g_t(x) = 1$ for a feasible schedule and the problem of minimizing tardiness may be expressed as

$$\min_{x \in X} \sum_{i=1}^{n} \max(x_i + p_i - d_i, 0) \tag{3.1a}$$

$$g_t(x) = 1, \qquad t = 1, \ldots, T. \tag{3.1b}$$

Introducing dual variables $u = (u_1, \ldots, u_T)$ associated with (3.1b) we define the Lagrangian function

$$L(x, u) = \sum_{i=1}^{n} \max(x_i + p_i - d_i, 0) + \sum_{t=1}^{T} u_t g_t(x)$$

$$= \sum_{i=1}^{n} \left(\max(x_i + p_i - d_i, 0) + \sum_{t=x_i+1}^{x_i+p_i} u_t \right)$$

and the Lagrangian problem

$$w(u) = \min_{x \in X} L(x, u) - \sum_{t=1}^{T} u_t.$$

It is well known that $w(u)$ lower bounds the optimal value of (3.1) for any u, a fact which is easily shown by letting $x^* \in X$ denote an optimal solution for (3.1) and

observing that

$$w(u) \leq \sum_{i=1}^{n} (x_i^* + p_i - d_i, 0) + \sum_{t=1}^{T} u_t(g_t(x^*) - 1) = \sum_{i=1}^{n} (x_i^* + p_i - d_i, 0).$$

The best lower bound is given by those u which solve the dual problem

$$\max_u w(u).$$

Note that if we regard u_t as a price charged for using the machine during the interval $[t-1, t]$ then the objective function of the Lagrangian problem has a very natural interpretation. The term for each job is simply a direct cost (the tardiness of the job) plus a charge u_t for machine usage for each period t during which the job is in process.

Lawler [18] has shown that a transportation problem may be used to obtain bounds for the one-machine scheduling problem. Reference [18] describes an allowable class of objective function coefficients for the transportation problem which give valid bounds. It may be shown that the optimal value of the dual problem given here dominates the transportation problem bound for any coefficients in the allowable class. In particular it dominates the Gelders–Kleindorfer [10] bound obtained from a specific set of coefficients.

We will first specify an algorithm for solving the Lagrangian problem $\min_{x \in X} L(x, u)$ with u fixed. Then we will discuss solution of the dual problem $\max_u w(u)$. The amount of computation required by the Lagrangian algorithm is proportional to nT. This algorithm and the other results of this section remain valid if $\max(x_i + p_i - d_i, 0)$ is replaced by any function of x_i. The algorithm is also applicable to the Lagrangian problem derived in [6] for the general resource-constrained network scheduling problem. We have found that the algorithm given here is much more efficient than the one first given in [6] for the Lagrangian problem and significantly reduced solution times have been obtained for the problems solved in [6] by using this algorithm. These results are reported in reference [7].

Let $\bar{B}_i$ denote the set of all predecessors of i, let $\hat{B}_i = \{i\} \cup \bar{B}_i$, and let

$$L_i(x_i, u) = \max(x_i + p_i - d_i, 0) + \sum_{t=x_i+1}^{x_i+p_i} u_t.$$

Note that

$$L(x, u) = \sum_{i=1}^{n} L_i(x_i, u).$$

The algorithm for the Lagrangian problem recursively computes a quantity $c_i(t)$ for all i and $t = 1, \ldots, T$. This quantity will turn out to give the optimal value of a partial Lagrangian problem containing all jobs j with $j \in \hat{B}_i$ and with the additional restriction that these jobs be completed by time t. The recursion for $c_i(t)$ is based on the following facts. If $x_i < t - p_i$ in an optimal solution to the partial Lagrangian problem then $c_i(t) = c_i(t-1)$. Otherwise $x_i = t - p_i$ in an optimal solution and $c_i(t)$ is equal to $L_i(t - p_i, u)$ plus the objective function contribution from jobs in $\bar{B}_i$. By definition $\bar{B}_i$ is equal to $\bigcup_{k \in B_i} \hat{B}_k$. In Lemma 3.1

below we show that the sets $\hat{B}_k$, $k \in B_i$ are pairwise disjoint in which case the contribution from jobs in $\bar{B}_i$ is just $\sum_{k\in B_i} c_k(t-p_i)$. The proof of Lemma 3.1 is precisely where we require $|A_i| \leqslant 1$.

Lemma 3.1. *If $j \notin \hat{B}_i$ and $i \notin \hat{B}_j$, then $\hat{B}_i \cap \hat{B}_j = \emptyset$.*

Proof. Suppose $k \in \hat{B}_j \cap \hat{B}_i$, $j \notin \hat{B}_i$ and $i \notin \hat{B}_j$. Then as a consequence of the definition of $\hat{B}_j$ and $\hat{B}_i$ there exists $j_1, \ldots, j_p$ and $i_1, \ldots, i_q$ such that $j_1 = i_1 = k$, $j_p = j$, $i_q = i$, and

$$j_1 \in B_{i_2}, \ldots, j_{p-1} \in B_{j_p},$$
$$i_1 \in B_{i_2}, \ldots, i_{q-1} \in B_{i_q}.$$

Let $\hat{l}$ be the largest value for which $j_l = i_l$, $l = 1, \ldots, \hat{l}$ and note that $1 \leqslant \hat{l} \leqslant \min(p, q)$. If $\hat{l} = \min(p, q)$ then either $j \in \hat{B}_i$ $(\hat{l} = p)$ or $i \in \hat{B}_j$ $(\hat{l} = q)$ which contradicts our hypothesis. If $\hat{l} < \min(p, q)$, then $|A_{i_{\hat{l}}}| \geqslant 2$ which contradicts $|A_i| \leqslant 1$ for all i. Hence $\hat{B}_j \cap \hat{B}_i = \emptyset$.

We will assume that jobs are indexed so that $B_i \subseteq \{1, \ldots, i-1\}$. It is easy to show that such an indexing exists given the fact that the graph defined by the sets B_i is acyclic.

Algorithm for $\min_{x\in X} L(x, u)$

Step 1. Initialize $i = 0$.

Step 2. Set $i = i + 1$,

$$c_i(t) = \begin{cases} \infty & t = 0, \ldots, a_i + p_i - 1, \\ \min\left\{c_i(t-1), L_i(t-p_i, u) + \sum_{k\in B_i} c_k(t-p_i)\right\} & \\ & t = a_i + p_i, \ldots, b_i + p_i, \\ c_i(b_i + p_i) & t = b_1 + p_i + 1, \ldots, T, \end{cases} \tag{3.2}$$

and

$$\gamma_i(t) = \begin{cases} 0 & \text{if } c_i(t) = c_i(t-1) \text{ or } t = 0, \\ 1 & \text{otherwise.} \end{cases}$$

Step 3. If $i = n$ go to 4. Otherwise go to 2.

Step 4. We now have $c_n(T) = \min_{x\in X} L(x, u)$. To obtain a solution x^* for which $L(x^*, u) = c_n(T)$ we first set $x_n^* = t^* - p_n$ where t^* is the largest value for which $\nu_n(t^*) = 1$. Then for $i = n-1, n-2, \ldots, 1$ set

$$\begin{aligned} x_i^* = \max \quad & t - p_i, \\ \text{s.t.} \quad & t \leqslant x_j^*, \quad j \in A_i, \\ & \gamma_i(t) = 1. \end{aligned}$$

Each of these problems is feasible because $\gamma_i(a_i + p_i) = 1$ for all i and hence $t = a_i + p_i$ is a feasible solution.

The procedure for computing $c_i(t)$ is essentially a dynamic programming scheme in which states correspond to the pair i, t for $i = 1, \ldots, n$ and $t = 0, \ldots, T$. A notable difference with usual *DP* recursions is that the optimal return at a state i, t is computed from the returns for several other states. This variation on dynamic programming has been termed nonserial dynamic programming. See [20, pp. 189–206] for a general discussion.

The operations of the algorithm consist entirely of addition, subtraction, and comparison. We will show that the computational requirements of the algorithm are proportional to nT or $n^2\bar{p}$ where $\bar{p} = T/n$ is the average job processing time. Clearly steps 1, 3 and 4 require at most nT time. The major parts of step 2 require the computation of $\Sigma_{\tau=t-p_i+1}^{t} u_\tau$ for the evaluation of $L_i(t - p_i, u)$ and the computation of $\Sigma_{k\in B_i} c_k(t - p_i)$. Once $\Sigma_{\tau=a_i+1}^{a_i+p_i} u_\tau$ is calculated then subsequent sums may be obtained recursively with a single addition and subtraction by

$$\sum_{\tau=t-p_i+1}^{t} u_\tau = \sum_{\tau=t-p_i}^{t-1} u_\tau + u_t - u_{t-p_i}$$

so that this part of step 2 requires computation proportional to nT. Since $|A_i| \leq 1$ for all i, $\Sigma_{i=1}^{n} |B_i| = \Sigma_{i=1}^{n} |A_i| \leq n$ and computation of $\Sigma_{k\in B_i} c_k(t - p_i)$ requires nT time.

The following two lemmas and theorem give a proof of optimality for the algorithm. This proof is not a prerequisite for the remainder of the paper. In Lemma 3.2 we prove that $c_i(t)$ is the optimal value of the partial Lagrangian problem described earlier.

Lemma 3.2. *For fixed u define the family of programs*

$$v_i(t) = \min \sum_{j\in\hat{B}_i} L_j(x_j, u), \tag{3.3i, t a}$$

$$\text{s.t.} \quad x_i + p_i \leq t, \tag{3.3i, t b}$$

$$\left.\begin{aligned} x_j &\geq x_l + p_l, \quad l \in B_j, \\ a_j &\leq x_j \leq b_j, \\ x_j &\text{ integer}, \end{aligned}\right\} \quad j \in \hat{B}_i \qquad \begin{matrix}(3.3i, t \text{ c}) \\ (3.3i, t \text{ d}) \\ (3.3i, t \text{ e})\end{matrix}$$

for $t = 0, \ldots, T$ and $i = 1, \ldots, n$ where by convention $v_i(t) = \infty$ if $(3.3i, t)$ is infeasible. Then $c_i(t) = v_i(t)$, $t = 0, \ldots, T$, $i = 1, \ldots, n$.

Proof. Note that for $t < a_i + p_i$ $(3.3i, t$ b) and $(3.3i, t$ d) for $j = i$ imply that $(3.3i, t)$ is infeasible and $v_i(t) = +\infty = c_i(t)$. For $t \geq a_i + p_i$ $(3.3i, t)$ is feasible and $v_i(t) > -\infty$.

We will establish the lemma for $t \geq a_i + p_i$ by induction. For any i we consider two cases, $a_i + p_i \leq t \leq b_i + p_i$, and $b_i + p_i < t \leq T$ and show that if $c_j(\tau) = v_j(\tau)$ for $\tau = 0, \ldots, t - 1$ and $j = 1, \ldots, n$, then $c_i(t) = v_i(t)$.

First consider $a_i + p_i \leq t \leq b_i + p$. Let $\bar{x}_j$, $j \in \hat{B}_i$ denote an optimal solution to $(3.3i, t)$. Then $v_i(t) = \Sigma_{j\in\hat{B}_i} L_j(\bar{x}_j, u)$.

By definition $\hat{B}_i = (\bigcup_{k\in B_i} \hat{B}_k) \cup \{i\}$. By Lemma 3.1 the sets $\hat{B}_k$ for $k \in B_i$ are pairwise disjoint so

$$\sum_{j\in\hat{B}_i} L_j(\bar{x}_j, u) = L_i(\bar{x}_i, u) + \sum_{k\in B_i} \sum_{j\in\hat{B}_k} L_j(\bar{x}_j, u).$$

First suppose $\bar{x}_i = t - p_i$. Then for any $k \in B_i$, $\bar{x}_k + p_k \leq t - p_i$ so $\bar{x}_j$, $j \in \hat{B}_k$ is feasible in $(3.3k, t - p_i)$. Thus by the induction hypothesis

$$c_k(t - p_i) \leq \sum_{j\in\hat{B}_k} L_j(\bar{x}_j, u).$$

If

$$c_k(t - p_i) < \sum_{j\in\hat{B}_k} L_j(\bar{x}_j, u)$$

for some $k \in B_i$, then there is an $\bar{\bar{x}}_j$, $j \in \hat{B}_k$ for which

$$\sum_{j\in\hat{B}_k} L_i(\bar{\bar{x}}_j, u) < \sum_{j\in\hat{B}_k} L_j(\bar{x}_j, u).$$

But then $\bar{x}_j$, $j \in \hat{B}_i \sim \hat{B}_k$, $\bar{\bar{x}}_j$, $j \in \hat{B}_k$ is feasible in $(3.3i, t)$ and

$$\sum_{j\in\hat{B}_i\sim\hat{B}_k} L_j(\bar{x}_j, u) + \sum_{j\in\hat{B}_k} L_j(\bar{\bar{x}}_j, u) < \sum_{j\in\hat{B}_i} L_j(\bar{x}_j, u)$$

which contradicts the optimality of $\bar{x}_j$. Hence

$$c_k(t - p_i) = \sum_{j\in\hat{B}_k} L_j(\bar{x}_j, u)$$

for all $k \in B_i$ and

$$v_i(t) = \sum_{j\in\hat{B}_i} L_j(\bar{x}_j, u) = L_i(t - p_i, u) + \sum_{k\in B_i} c_k(t - p_i).$$

Since any solution feasible in $(3.3i, t - 1)$ is also feasible in $(3.3i, t)$ we must have $v_i(t) \leq v_i(t - 1)$. Then since $v_i(t - 1) = c_i(t - 1)$ by the induction hypothesis and $c_i(t) = \min(c_i(t - 1), L_i(t - p_i, u) + \Sigma_{k\in B_i} c_k(t - p_k)$, the desired result $v_i(t) = c_i(t)$ follows when $\bar{x}_i = t - p_i$.

If $\bar{x}_i < t - p_i$ then $\bar{x}_j$, $j \in \hat{B}_i$ is feasible and hence optimal in $(3.3i, t - 1)$. Thus $v_i(t) = v_i(t - 1) = c_i(t - 1)$. Since $v_i(t)$ is the optimal value of $(3.3i, t)$ it must satisfy $v_i(t) \leq L_i(t - p_i, u) + \Sigma_{k\in B_i} c_k(t - p_k)$ and $c_i(t) = c_i(t - 1) = v_i(t)$.

Finally, for $b_i + p_i < t \leq T$, constraint (3.3i, t b) is superfluous with (3.3i, t d) for $j = i$ so $(3.3i, t)$ and $(3.3i, b_i + p_i)$ are identical programs. Hence $v_i(t) = v_i(b_i + p_i) = c_i(b_i + p_i) = c_i(t)$.

The following lemma relates x^* to the functions $c_i(t)$. We then present a theorem which establishes the optimality of x^* for the problem $\min_{x\in X} L(x, u)$.

Lemma 3.3. *For* $i = 1, \ldots, n$

$$c_i(x_i^* + p_i) = \sum_{j\in\hat{B}_i} L_j(x_j^*, u).$$

Proof. The proof will use induction on $|\hat{B}_i|$. First consider i such that $|\hat{B}_i| = 1$. In this case $B_i = \emptyset$. The form of step 4 of the algorithm implies $\gamma_i(x_i^* + p_i) = 1$ which, by step 2 and $B_i = \emptyset$ implies $c_i(x_i^* + p_i) = L_i(x_i^*, u)$.

Now we suppose for some $k \geq 1$ that the lemma is true for all i with $|\hat{B}_i| \leq k$ and show that it is true for any i for which $|\hat{B}_i| = k + 1$. The form of step 4 implies $\gamma_i(x_i^* + p_i) = 1$ which then implies by step 2 that $c_i(x_i^* + p_i) = L_i(x_i^*, u) + \Sigma_{j \in B_i} c_j(x_i^*)$. Again by step 4 and the fact that $i \in A_j$ for $j \in B_i$ we have $c_j(x_i^*) = c_j(x_i^* - 1) = \cdots = c_j(x_j^* + p_j)$. Then the lemma follows from the induction hypothesis.

Theorem 3.4. $L(x^*, u) = \min_{x \in X} L(x, u)$.

Proof. Let $K = \{k: A_k = \emptyset\}$. By the form of step 4 and fact that $A_k = \emptyset$ we have for $k \in K$, $c_k(T) = c_k(T-1) = \cdots = c_k(x_k^* + p_k)$. Then by Lemma 3.3, $c_k(T) = \Sigma_{i \in \hat{B}_k} L_i(x_i^*, u)$. By Lemma 3.2 $x_i^*, i \in \hat{B}_k$ is optimal in $(3.3k, T)$.

Clearly $\bigcup_{k \in K} \hat{B}_k = \{1, \ldots, n\}$ and by Lemma 3.1 the sets $\hat{B}_k$ for $k \in K$ are pairwise disjoint. These facts imply that the set X is the Cartesian product of the $|K|$ sets of x_i, $i \in \hat{B}_k$ which satisfy the constraints of $(3.3k, T)$, $k \in K$. Since the objective is separable in x_i and x_i^*, $i \in \hat{B}_k$ is optimal in $(3.3k, T)$, an optimal solution to $\min_{x \in X} L(x, u)$ is given by the x_i^*, $i \in B_k$ for $k \in K$.

Now consider the dual problem $\max_u w(u)$. For this problem we will use a subgradient method described in [15] and further refined in [16]. Let u^0 be any initial value for u, for example $u^0 = 0$. The method generates a sequence $\{u^k\}$ by the rule

$$u^{k+1} = u^k + \lambda_k \frac{(z^* - w(u^k))(g(x^k) - e)}{\|g(x^k) - e\|^2}, \qquad k = 0, 1, \ldots$$

where z^* is the value of a known feasible solution to (3.1), λ_k is a scalar satisfying $0 < \lambda_k \leq 2$, x^k is the optimal solution to $\min_{x \in X} L(x, u^k)$ obtained with the Lagrangian algorithm previously given, $\|\cdot\|$ denotes the Euclidean norm and e is a T component vector of ones. We initialize z^* using the feasible solution determined with a sequencing heuristic given in Section 4. We also obtain a feasible solution at each iteration by sequencing jobs in order of increasing x_i^k. The target z^* is updated whenever the objective value of this solution is less than z^*.

For the problems solved here $u^0 = 0$ and $\lambda_0 = 2$ were used. The sequence λ_k was generated as follows. For $k \geq 1$, $\lambda_k = \lambda_{k-1}$ was used, unless $w(u)$ had failed to increase for 5 consecutive iterations, that is unless $w(u^j) \leq w(u^{k-5})$ $j = k-5, \ldots, k-1$. Then we set $\lambda_k = \frac{1}{2}\lambda_{k-1}$.

Unless we discover an x^k with $g(x^k) = e$, or u such that $w(u) = z^*$, there is no means of knowing if the subgradient method has found an optimal solution to the dual. This is not a practical problem in our work however. We simply perform a fixed number of iterations and then use the current value of $w(u)$ as our lower bound. Details on the number of iterations used are given in Section 4.

We conclude this section with three observations. The first concerns a

structural property of the dual problem which has been important computationally. It is easy to see that $\Sigma_{t=1}^{T} g_t(x) = T$ for all $x \in X$, a fact which can be used to show by direct algebraic manipulation that $L(x, u + \lambda e) - \Sigma_{t=1}^{T}(u_t + \lambda) = L(x, u) - \Sigma_{t=1}^{T} u_t$ and hence $w(u + \lambda e) = w(u)$. This means that a normalization constraint may be imposed on u in solving the dual. Such a normalization is inherent in the subgradient method since values of u determined by the method satisfy

$$\sum_{t=1}^{T} u_t^{k+1} = \sum_{t=1}^{T} u_t^k + \lambda_k \frac{(z^* - w(u^k))}{\|g(x^k) - e\|^2} \sum_{t=1}^{T} (g_t(x^k) - 1) = \sum_{t=1}^{T} u_t^k.$$

Reasoning inductively we have $\Sigma_{t=1}^{T} u_t^{k+1} = \Sigma_{t=1}^{T} u_t^0$. Since we have taken $u^0 = 0$ the normalization used here is $\Sigma_{t=1}^{T} u_t = 0$.

The constraint $u \geq 0$ is another valid normalization, one which is natural since $\Sigma_{t=1}^{T} g_t(x) = T$ implies that it is valid to replace the equality constraints (3.1b) by inequalities. This constraint was used in our early computational work until we discovered that the current dual requires significantly fewer iterations to obtain the same lower bound.

This fact is not surprising. Convergence results given in [16] for the subgradient method are based on the fact if λ_k is sufficiently small then u^{k+1} will be closer to the optimal set than u^k, where the distance from a point u^k to a closed set U is $\min_{u \in U} \|u - u^k\|$. It may be shown that unless $u = 0$ is optimal in the dual, the initial value $u^0 = 0$ is strictly closer to the set of optimal solutions satisfying $\Sigma_{t=1}^{T} u_t = 0$ than to those satisfying $u \geq 0$. Let $\bar{u}$ be any optimal solution satisfying $\bar{u} \geq 0$ and let $K = (\Sigma_{t=1}^{T} \bar{u}_t)/T$. Then $\bar{\bar{u}} = \bar{u} - Ke$ is another optimal solution satisfying $\Sigma_{t=1}^{T} \bar{\bar{u}} = 0$. Further

$$\begin{aligned} \|\bar{\bar{u}} - 0\|^2 &= \sum_{t=1}^{T} (\bar{u}_t - K)^2 \\ &= \sum_{t=1}^{T} (\bar{u}_t)^2 - 2K \sum_{t=1}^{T} \bar{u}_t + TK^2 \\ &= \|\bar{u} - 0\|^2 - TK^2 \end{aligned}$$

and unless $\bar{u} = 0$, $K > 0$. This fact suggests that if a choice exists, equality constraints are preferable to inequality constraints for the application of dual methods to combinatorial problems.

Our second observation concerns the computation required by the Lagrangian algorithm. We have shown that this computation is proportional to $n^2\bar{p}$, where $\bar{p}$ is the average job processing time. If $\bar{p}$ is large, the Lagrangian problem may be difficult to solve even if n is small. This difficulty can be avoided by simply ignoring the constraint $g_t(x) = 1$ for some periods of the problem. If large process times are a problem we suggest replacing the dual problem by

$$\begin{aligned} \max \quad & w(u), \\ \text{s.t.} \quad & u \geq 0, \\ & u_t = 0, \quad t/k \text{ not integer} \end{aligned}$$

where k is an integer greater than 1. The time periods ignored are those for which

u_t is required to be zero. While addition of the constraints $u_t = 0$ for t/k not integer may reduce the maximum value of $w(u)$, it also simplifies solution of the Lagrangian problem. We can modify step 2 of the algorithm for the Lagrangian problem by setting $c_i(t) = c_i(t-1)$ for all t such that $(t - p_i)/k$ is not integer. This is because $u_{t-p_i} = 0$ and

$$\sum_{\tau=t-p_i+1}^{t} u_\tau = \sum_{\tau=t-p_i}^{t-1} u_\tau - u_{t-p_i} + u_t = \sum_{\tau=t-p_i}^{t-1} u_\tau + u_t \geqslant \sum_{\tau=t-p_i}^{t-1} u_\tau.$$

Hence

$$\begin{aligned} c_i(t-1) &\leqslant \max(t - d_i - 1, 0) + \sum_{\tau=t-p_i}^{t-1} u_\tau \\ &\leqslant \max(t - d_i, 0) + \sum_{\tau=t-p_i+1}^{t} u_\tau. \end{aligned}$$

Under this modification the computation for the Lagrangian algorithm is proportional to $n^2\bar{p}/k$.

Some minor modifications to the algorithm for the dual are necessitated by the additional constraints. Obviously the formula for updating u_t is applied only if t/k is integer. If the updated u_t would be negative, the value 0 is used instead.

Finally, we would like to note some similarities between the recursion for computing $c_i(t)$ and a standard dynamic programming recursion for the knapsack problem given on page 217 of [9]. If the size of the knapsack is equal to the parameter T used here and we regard jobs as variables in the knapsack problem then both recursions evaluate a function which gives the optimal value of a family of knapsack problems for knapsack sizes between 0 and T and for n different subsets of the variables (jobs). This function is given as the minimum of two terms formed from the optimal values of other knapsack problems and a direct cost. In each case the number of steps required to solve the problem is proportional to the product of the number of variables (jobs) and the size of the knapsack.

The two problems are not however identical. Both the differences and similarities may be highlighted by examination of the Lagrangian problem when the successor sets A_i have a special form. Suppose for $k < n$ that $B_1 = A_k = \emptyset$ and $B_i = \{i-1\}$, $i = 2, \ldots, k$. Then jobs $1, \ldots, k$ correspond to a separate component of the precedence graph and the Lagrangian optimization may be performed independently for them. If we let $y_1 = x_1$, $y_i = x_i - x_{i-1} - p_{i-1}$, $i = 2, \ldots, k$, $y = (y_1, \ldots, y_k)$, $b = T - \Sigma_{i=1}^{k} p_i$, and

$$f(y) = \sum_{i=1}^{k} L_i\left(\sum_{j=1}^{i} y_j + \sum_{j=1}^{i-1} p_j, u\right)$$

then the Lagrangian problem for jobs $1, \ldots, k$ is the following knapsack problem with nonlinear objective,

$$\min f(y),$$

$$\sum_{i=1}^{k} y_i \leqslant b,$$

$$y_i \geqslant 0 \text{ and integer.}$$

We note that the recursion given in [9] is not applicable to this problem because of the nonlinear objective.

It is interesting to note the appearance of an embedded knapsack problem in the analysis of other combinatorial problems. Gilmore and Gomory [12] use the standard knapsack problem in their algorithm for the cutting back stock problem and Fisher and Shapiro [8] have used the group knapsack problem devised by Gomory to form a dual problem for the general integer programming problem. These analyses share some interesting strategic similarities. For example, in each case difficulties can arise if the knapsack problem is too large. For the general IP problem, the size of the knapsack is the absolute value of the LP basis determinant and can be large if the constraint coefficients are large. Gorry, Shapiro, and Wolsey [13] have suggested a method for dealing with this difficulty that depends on relaxing inequality constraints by dividing the coefficients and right-hand sides by a constant greater than one and rounding in the appropriate direction. Here the size of the knapsack equals the sum of the job process times and can be disproportionately large if the process times are given as large numbers. In this case we suggested specifying a suitable integer k and ignoring all time periods which are not devisable by k. It can be shown that this is essentially the same as dividing all process times by k and rounding down, in which case this scheme is surprisingly close to that of Gorry, Shapiro, and Wolsey.

4. Branch and bound algorithm

In the branching scheme we use each node corresponds to a specific sequence for some subset of the n jobs which are required to be performed last. Specifically, let $K = \{k_1, \ldots, k_l\}$ denote an ordered set of l distinct integers selected from $\{1, \ldots, n\}$ to satisfy the property $A_{k_1} = \emptyset$ and $A_{k_i} \subseteq \{k_1, \ldots, k_{i-1}\}$ for $i = 2, \ldots, l$. Let

$$X_K = \{x \in X: x_{k_1} = T - p_{k_1}, x_{k_2} = T - p_{k_1} - p_{k_2}, \ldots, x_{k_l} = T - \sum_{j=1}^{l} p_{ij},$$

$$\text{and} \quad x_j \leqslant x_{k_i} - p_j, j \notin K\}.$$

Nodes will correspond to sets K and at node K we require $x \in X_K$.

The fundamental step of the algorithm selects a node K and attempts to either find an optimal solution over the set X_K or show that X_K may be ignored in the search for an optimal solution. If this can be done we say the node K is fathomed. If we fail in the attempt to fathom a node then we branch, that is form new nodes from node K by partitioning the set X_K into subsets. The new nodes are of the form $\{K, i\}$ for all i with $A_i \subseteq K$ but $i \notin K$.

There are three devices for fathoming nodes. First, for any u a lower bound on the optimal value of (3.1) with $x \in X_K$ is given by $w_K(u) = \min_{x \in X_k} L(x, u) - \Sigma_{t=1}^{T} u_t$. This problem may be solved by the Lagrangian algorithm in Section 3. Let z^*

denote the objective value of a known feasible solution for (3.1). If for some u, $w_K(u) \geq z^*$, then x_K cannot contain a solution better than the known one and the node K is fathomed by bound. (Actually, since d_i and p_i are integers the objective value of any solution must be integer and if $w_K(u)$ is fractional, the next largest integer is a valid lower bound. To allow for computational error in the calculation of $w_K(u)$ the computer code uses the largest integer not greater than $w_K(u)+0.9$ in place of $w_K(u)$.)

Let $x_K(u)$ denote the solution to $\min_{x \in X_k} L(u, x)$ obtained by the Lagrangian algorithm. Note that $x_K(u)$ is feasible in (3.1) if and only if $g_t(x_K(u)) = 1$ for $t = 1, \ldots, T$. If this occurs then $x_K(u)$ is optimal in (3.1) under the restriction $x \in X_K$ and node K is also fathomed.

A third device for fathoming nodes is based on the principle of optimality of dynamic programming. For a node $K = \{i_1, \ldots, i_l\}$ let $\Pi(K) = \sum_{k=1}^{l} \max(0, T - \sum_{j=1}^{k-1} p_{i_j} - d_{i_k})$ denote the objective value for the jobs of K performed last in the sequence $i_l, \ldots, i_1$. Suppose that two nodes of the branch and bound tree correspond to sets K_1 and K_2 which contain the same jobs and that $\Pi(K_2) \geq \Pi(K_1)$. Then no solution in X_{K_2} can have a lower objective value than an optimal solution in X_{K_1} and the node K_2 may be ignored. In this case we will say node K_2 is dominated by node K_1. This is exactly the reasoning used by the dynamic programming algorithms in [14] and [24] for the one-machine scheduling problem. The use of this result here amounts to a synthesis of dynamic programming and branch and bound. See [19] for a general discussion of this idea.

It is possible that reapplication of Theorem 2.1 could be fruitful at a particular node. This is because the requirement $x \in X_K$ imposes new conditions which may cause the test of the theorem to hold. This possibility is not included in the current version of the algorithm.

Earlier in our work we used a branching scheme in which nodes corresponded to sequences of jobs to be performed *first*. While the current scheme is on average dramatically better than the earlier one, there have been a few problems where it required more than twice as much solution time. Both schemes would undoubtedly be dominated by a mixed strategy in which nodes correspond to partial sequences at both ends of the schedule. When branching the decision of whether to fix a job at the start or end of the current unscheduled set could be based on the increase in the lower bound for each case in the same way that a branch variable is selected in integer programming algorithms.

A feasible solution with which to initialize z^* was obtained by the following procedure. Let $S = \{k_1, \ldots, k_l\}$ denote a sequence for a set of jobs which have been scheduled to be performed first and let $U = \{1, \ldots, n\} \sim S$ denote the unscheduled set. Begin with $S = \emptyset$ and augment S according to the following rule. Let $U' = \{i \in U: B_i \subseteq S\}$, $t = \sum_{i \in S} p_i$ and $t' = \sum_{i \in U'} p_i$. For each $i \in U'$ we compute a "probability" PT_i that job i will be tardy if not scheduled next. Set

$$\mathrm{PT}_i = \begin{cases} 1, & d_i \leq p_i + t, \\ \dfrac{t' + t - d_i}{t' - p_i}, & t + p_i < d_i < t + t'. \\ 0, & t + t' < d_i. \end{cases}$$

Let $\bar{t} \in U'$ be any job with maximum value of PT_i/p_i and set $S = \{k_1, \ldots, k_l, \bar{t}\}$. This procedure is an adaptation of Caroll's heuristic sequencing rule for general job shops [3]. Once a feasible schedule has been obtained in this way adjacent pairwise interchanges are attempted until no improvement can be made. This determines an initial value for z^*. We also update z^* whenever an improved feasible solution is discovered in the course of solving a Lagrangian problem in the branch and bound algorithm.

The flow chart in Fig. 2 gives a detailed statement of the branch and bound algorithm. The parameters I_1, I_2, I_3, shown are input quantities which specify the maximum number of iterations of the subgradient method to be performed at various points in the algorithm. The four major steps of the algorithm are numbered 1–4. Other steps are concerned with minor bookkeeping activities. Steps 1 and 2 attempt to fathom a selected node K by dominance and by bound. If node K is not fathomed then we go to step 3 and create new nodes. When a node $\{K, i\}$ is created it is tagged with an initial lower bound value $z_{K,i}$. If possible the next node to be fathomed is selected from those newly created and optimization

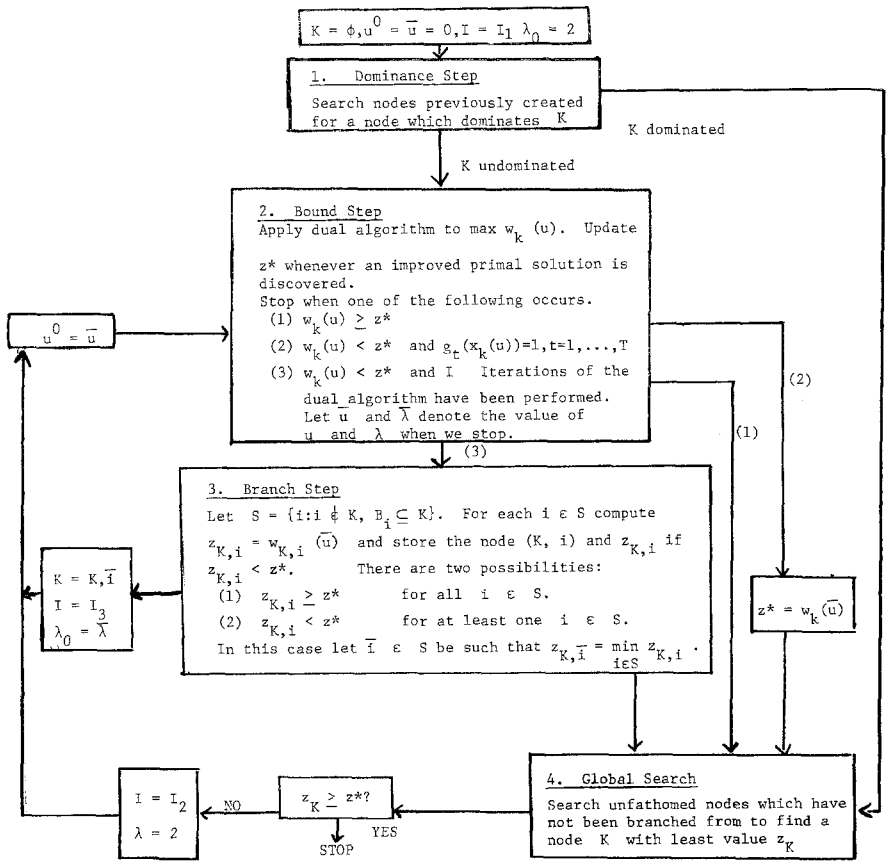

Fig. 2. Flowchart of the branch and bound algorithm.

of the new dual is continued with the step size λ_k at its current value. Otherwise a next node is selected by a global search in step 4. We terminate when this search fails to produce an unfathomed node.

To illustrate this algorithm we give in Fig. 3 the complete enumeration tree for the 50-job example given in Sections 1 and 2. The number inside of a node indicates the order in which that node was selected for attempted fathoming. The number beside a branch indicates the job sequenced next along that branch. Numbers beside a node give the lower bound obtained when the node is created. If no value is given, the lower bound is the same as at the predecessor node. When two numbers such as 1462/1464 are given, the upper number is the lower bound obtained when the node was created while the lower number is an improved lower bound obtained by optimizing the dual problem when the node was selected for attempted fathoming. A number in a square beneath a node like [4] indicates that

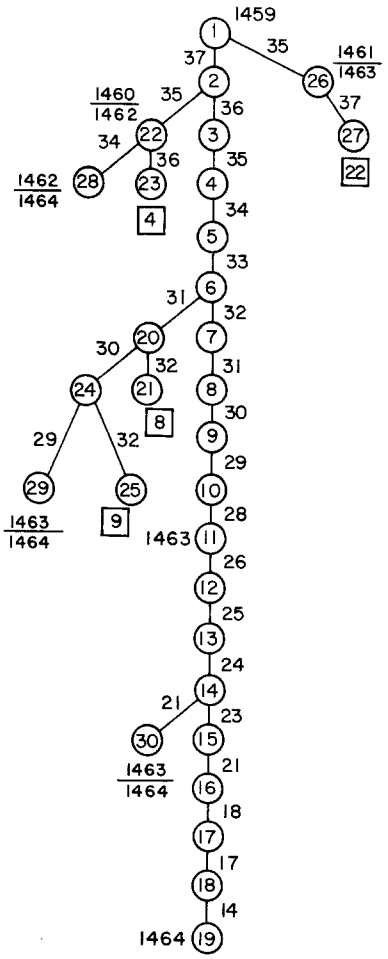

Fig. 3. Enumeration tree for the example.

the node is dominated by node 4. The enumeration begins with a feasible solution with objective value 1464. This solution was found in the initial optimization of the dual at node 1 and was in this case optimal.

5. Computational experience

The algorithm has been applied to a sample of 75 randomly generated test problems. Process times for these problems were obtained from a uniform distribution on the integers between 1 and 10. Once process times for a problem had been obtained $T = \sum_{i=1}^{n} p_i$ was computed and due dates were selected from a distribution which depends on T and two parameters R and τ called due date range and tardiness factor. The distribution is uniform over the integers between $T(1-\tau-R/2)$ and $T(1-\tau+R/2)$, where RT is the range of the distribution and $T(1-\tau)$ is the mean. In a certain average sense τ is the fraction of jobs which are late in an optimal solution. More precisely if all process times are equal and all due dates are equal to the distribution mean of $T(1-\tau)$ then τ is the fraction of jobs which are late in an optimal solution. A number of authors ([1], [21], [24]) have noted that the values of τ and R have a strong effect on problem difficulty and that problems obtained from distributions with τ between 0.6 and 0.8 and with $R = 0.2$ are the most difficult.

The 75 problems considered here were divided equally over three sizes, $n = 20$, 30, and 50. Each set of 25 problems of a particular size was distributed over four pairs of values for τ and R. There were 10 problems with $\tau = 0.5$ and $R = 1.0$, and 5 problems each for the pairs $\tau = 0.8$, $R = 0.4$, $\tau = 0.8$, $R = 0.2$, and $\tau = 0.65$, $R = 0.2$. Thus the sample included a number of very large and difficult problems. Data for all problems is available from the author on request.

Extensive information for each problem solved is given in Tables 1, 2 and 3. Columns (1) and (2) in each table give the value of τ and R. Column (3) gives the number of jobs remaining in the problem after the reduction described in Section 2 was applied. Column (4) gives the optimal objective function value for the original unreduced problem. Columns (5)–(9) all deal with the reduced problem and report various objective function values. Column (5) gives the value of the feasible solution obtained by the heuristic sequencing rule described in Section 4. As discussed in Section 3, each time we solve a Lagrangian problem we obtain a feasible solution by sequencing jobs in order of their x_i values. Column (6) gives the value of the best improved feasible solution (if any) discovered by this means during the optimization of the dual problem at the first node of the branch and bound tree. Obtaining this solution requires a comparatively small amount of computation. Column (7) gives the optimal value for the reduced problem. Column (8) gives the lower bound obtained at the first node of the tree with the initial application of the dual algorithm. Finally, Column (9) gives the lower bound for $u = 0$, the value with which we begin solution of the dual. Column (10) gives the number of nodes in the enumeration tree at which fathoming was attempted. Column (11) gives the cumulative number of iterations of the dual algorithm which were executed in all fathoming attempts. This is the number of times a Lagrangian

Table 1
Results for the 20 job problems

(1) τ	(2) R	(3) Reduced problem size	(4) Optimal value complete problem	(5) z^*	(6) Initial feasible value	(7) Optimal value reduced problem	(8) $w(u^*)$	(9) $w(0)$	(10) Nodes	(11) Iter.	(12) 360-67 Seconds
0.5	1.0	14	76	85	77	76	74	9	5	73	4.98
0.5	1.0	3	109	27		27	27	10	1	9	0.55
0.5	1.0	0	263								0.28
0.5	1.0	0	66								0.28
0.5	1.0	0	16								0.28
0.5	1.0	15	205	136	135	134	128	37	18	146	10.19
0.5	1.0	7	36	27	25	25	25	0	1	20	1.42
0.5	1.0	13	41	40		40	40	21	1	7	0.76
0.5	1.0	9	174	93		93	92	54	7	78	2.81
0.5	1.0	0	32								0.28
0.8	0.4	0	491								0.28
0.8	0.4	15	395	143	141	141	141	78	1	38	3.06
0.8	0.4	0	630								0.28
0.8	0.4	5	566	19	18	18	18	11	1	26	0.88
0.8	0.4	4	533	17		17	17	10	1	12	0.61
0.8	0.2	15	390	141	136	135	135	60	4	62	4.62
0.8	0.2	0	655								0.73
0.8	0.2	13	419	113	96	96	96	38	1	32	2.72
0.8	0.2	20	574	615	574	574	574	468	1	43	5.11
0.8	0.2	12	593	128	122	121	121	55	8	88	4.92
0.65	0.2	13	356	89	88	85	82	0	9	110	7.17
0.65	0.2	15	412	252		252	252	131	1	36	2.94
0.65	0.2	15	318	141	131	131	131	0	1	39	4.72
0.65	0.2	15	359	172	169	168	168	42	2	52	5.61
0.65	0.2	13	271	108		99	97	0	4	57	5.00

Average solution time: 2.82 seconds.

problem was solved. Most of the computation time is spent solving these problems. Column (12) gives the total solution time for each problem.

To simplify the computer programming arcs of the precedence networks were removed until $|A_i| \leq 1$ and $|B_i| \leq 1$ for all i was satisfied. Values of the parameters I_1, I_2 and I_3 specified in Section 4 were given by $I_3 = 5$, $I_2 = 30$, and $I_1 = 50$ if the reduced problem contained no more than 25 jobs, $I_1 = 100$ otherwise. Selection of these values was somewhat arbitrary. The way in which starting solutions are obtained for the dual problem at various points in the branch and bound procedure led us to believe that the values should satisfy $I_1 > I_2 > I_3$. The specific selection was based largely on observations made while debugging the computer program. It is quite possible that further study of the selection of these values would improve algorithm performance. We believe that $w(u^*)$ is quite close to the optimal value of the dual, although in some cases, particularly when $I_1 = 50$, we know that additional iterations of the subgradient method would produce an improved bound.

To obtain additional information about how closely $w(u^*)$ approached the dual optimal value we used the BOXSTEP Method of Hogan, Marsten and Blanken-

Table 2
Results for the 30-job problems

(1) τ	(2) R	(3) Reduced problem size	(4) Optimal value complete problem	(5) z^*	(6) Initial feasible value	(7) Optimal value reduced problem	(8) $w(u^*)$	(9) $w(0)$	(10) Nodes	(11) Iter.	(12) 360-67 Seconds
0.5	1.0	21	160	110		110	110	56	1	19	2.44
0.5	1.0	0	22								0.96
0.5	1.0	25	269	252		251	248	162	37	294	22.29
0.5	1.0	0	61								0.96
0.5	1.0	0	45								0.96
0.5	1.0	11	223	143		143	143	120	1	8	1.46
0.5	1.0	20	127	49		49	48	14	2	53	5.44
0.5	1.0	23	172	136		136	136	70	1	32	3.69
0.5	1.0	0	176								0.96
0.5	1.0	15	202	77		77	77	15	1	22	2.51
0.8	0.4	17	917	104	103	103	102	30	3	60	5.73
0.8	0.4	19	709	95		94	92	30	13	109	9.28
0.8	0.4	18	1074	191	182	180	179	88	6	74	7.35
0.8	0.4	7	1102	32		32	32	20	1	2	1.29
0.8	0.4	14	853	85		85	83	38	10	89	5.22
0.8	0.2	19	1031	321	315	315	315	179	3	56	6.37
0.8	0.2	21	1169	430	423	423	423	265	1	39	6.52
0.8	0.2	19	859	275	246	246	246	178	1	40	4.25
0.8	0.2	25	879	690	651	649	645	494	20	177	19.06
0.8	0.2	13	712	19		19	18	0	8	82	3.62
0.65	0.2	21	612	640	613	612	609	439	17	148	14.75
0.65	0.2	24	615	440	428	428	428	242	7	75	11.91
0.65	0.2	22	964	611		610	608	313	20	211	30.86
0.65	0.2	28	511	535	515	511	511	257	8	135	25.29
0.65	0.2	22	662	272	253	252	246	51	42	432	58.75

Average solution time: 10.08 seconds.

ship [17] to obtain the precise optimal value of the dual for the problem reported in line 9 of Table 1. Since we know the optimal value of the problem is integer, lower bounds may be rounded up to the next largest integer and this is done in the values reported in column (18). The fractional value of $w(u^*)$ for this problem was 91.198. In our most successful experiment with BOXSTEP we first ran the subgradient method for 300 iterations to obtain a value of 91.968. Then BOXSTEP was applied and discovered an optimal solution to the dual with an objective value of 92.000. It is noteworthy that obtaining the precise optimal value of the dual using BOXSTEP required over 2 minutes on a 360-67.

A number of patterns exist in the computational results. The difference between the optimal value and $w(u^*)$ is consistently small. The largest gap of 274-263 occurred for the fifth 50-job problem. The bound $w(u^*)$ was actually equal to the optimal value for 31 of the 63 problems which were not completely solved by reduction. The difference between the first feasible solution obtained and the optimal value is also consistently small. The largest gap was 2590-2575 for the 20th 50-job problem. This difference was essentially independent of problem size and difficulty. The first solution was optimal in all but 23 cases. Since the first feasible

Table 3
Results for the 50-job problems

(1) τ	(2) R	(3) Reduced problem size	(4) Optimal value complete problem	(5) z^*	(6) Initial feasible value	(7) Optimal value reduced problem	(8) $w(u^*)$	(9) $w(0)$	(10) Nodes	(11) Iter.	(12) 360-67 Seconds
0.5	1.0	30	218	213		213	213	124	1	34	7.76
0.5	1.0	28	624	430	428	428	428	380	1	57	8.22
0.5	1.0	41	220	176		176	174	55	14	190	39.49
0.5	1.0	0	30								3.39
0.5	1.0	48	279	281		274	263	135	126	771	139.51
0.5	1.0	5	56	13		13	13	0	1	4	3.66
0.5	1.0	10	77	58	47	47	47	0	1	26	2.89
0.5	1.0	27	808	312		306	303	192	40	349	35.75
0.5	1.0	27	368	280		279	278	169	6	125	18.94
0.5	1.0	3	53	6		6	6	1	1	6	3.60
0.8	0.4	21	2690	158	157	157	155	125	4	66	5.56
0.8	0.4	9	3618	68		68	68	36	1	32	3.92
0.8	0.4	25	2195	122	113	113	112	11	3	67	10.05
0.8	0.4	40	3251	1743		1736	1729	1404	89	838	166.84
0.8	0.4	25	2342	177	164	164	164	60	1	42	8.89
0.8	0.2	33	3175	987	937	937	935	392	50	439	75.26
0.8	0.2	43	1620	1583	1569	1568	1564	821	28	269	84.16
0.8	0.2	40	2793	1567	1467	1465	1463	965	48	532	108.44
0.8	0.2	47	2673	2770	2673	2673	2670	2153	46	410	93.18
0.8	0.2	45	3023	2618	2590	2575	2570	2066	81	642	123.50
0.65	0.2	46	1733	1447	1404	1399	1390	515	91	955	401.62
0.65	0.2	37	1773	1176	1142	1142	1142	489	1	61	27.26
0.65	0.2	37	1540	1540	1464	1464	1459	668	30	360	101.16
0.65	0.2	29	1434	887	867	866	863	514	29	343	56.55
0.65	0.2	32	2222	945	885	885	884	431	19	233	57.63

Average solution time: 63.49 seconds.

solution is obtained relatively quickly the algorithm could be used quite practically to obtain approximately optimal solutions even for very large problems. The sharp lower bound $w(u^*)$ would be useful in providing an indication of the quality of the approximate solution obtained. The value of the first feasible solution can be compared with z^*, the value of the initial solution obtained by a heuristic. While this heuristic often produced good solutions, its performance was adversely affected by problem size and difficulty. This point should be noted in the empirical study of heuristics as they are often tested on small problems.

The enumeration trees obtained were generally small. With only one exception they contained less than 100 nodes. This is apparently a consequence of the sharp lower bounds and good feasible solutions.

Although the relation of problem difficulty to the parameters τ and R agree with the observations of Srinivasan [24], Baker and Martin [1], and Rinnooy Kan et al. [21], the disparity between "easy" and "hard" problems is less for our algorithm than for theirs.

A comparison of our results with that of others is difficult because other researchers have worked with small problems. Baker and Martin [1] tested

various algorithms on problems with n between 8 and 15. Average solution times for Srinivasan's algorithm, the best of those tested, appeared to double when n increased by 2. Approximately 1 second on a 360-65 was required to solve an average 15-job problem. If this experience were extrapolated, it would imply average solution times of 5.6 seconds, 179.2 seconds, and just over two days respectively for problems with 20, 30, and 50 jobs. The collection of values for τ and R in the Baker–Martin sample of test problems is comparable in difficulty to the one used here, although the average process times were larger than in ours. This may have increased solution times for Srinivasan's algorithm, although his algorithm does not have the direct dependence on average process time of ours.

The algorithm of Rinnooy Kan et al. [21] is also applicable to this problem and has performed quite well on a group of 10-, 15-, and 20-job weighted tardiness problems. This algorithm was recently applied to the 75 test problems used here. Generally solution times were about equal to ours for the 20-job problems but larger for the 30- and 50-job problems. In particular, one 30-job and several 50-job problems were not solved in 5 minutes of computing [22]. The longer solution times for the larger problems seems to be a consequence of somewhat weaker bounds. The comparison is partially obscured because the algorithm is programmed in ALGOL and was run on a Control Data Cyber 73-38.

Acknowledgment

I would like to express my appreciation to William Griffith and George Nemhauser for a number of helpful comments. Roy Marsten generously made his SEXOP subroutines available for one phase of the computational work.

References

[1] K.R. Baker and J.B. Martin, "An experimental comparison of solution algorithms for the single-machine tardiness problem", *Naval Research Logistics Quarterly* 21 (1) (1974) 187–199.
[2] K.R. Baker, *Introduction to sequencing and scheduling* (Wiley, New York, 1974).
[3] D.C. Carroll, "Heuristic sequencing of single and multi-component orders", Ph.D. dissertation, Massachusetts Institute of Technology (June 1965).
[4] S.E. Elmaghraby, "The one machine sequencing problem with delay costs", *Journal of Industrial Engineering* 17 (2) (1968).
[5] H. Emmons, "One machine sequencing to minimize certain functions of job tardiness", *Operations Research* 17 (4) (1969).
[6] M.L. Fisher, "Optimal solution of scheduling problems using Lagrange multipliers: Part I" *Operations Research* 21 (5) (1973) 1114–1127.
[7] M.L. Fisher, W. Northup and J.F. Shapiro, "Constructive duality in discrete optimization", *Mathematical Programming*, to appear.
[8] M.L. Fisher and J.F. Shapiro, "Constructive duality in integer programming", *SIAM Journal on Applied Mathematics* 27 (1) (1974).
[9] R.S. Garfinkel and G.L. Nemhauser, *Integer programming* (Wiley, New York, 1972).
[10] L. Gelders and P.R. Kleindorfer, "Coordinating aggregate and detailed scheduling decisions in the one-machine job shop: Part I, theory", *Operations Research* 22 (1974) 46–60.

[11] A.M. Geoffrion, "Lagrangian relaxation for integer programming", *Mathematical Programming Study* 2 (1974) 82–114.
[12] P.C. Gilmore and R.E. Gomory, "A linear programming approach to the cutting-stock problem", *Operations Research* 19 (1961) 849–859.
[13] G.A. Gorry, J.F. Shapiro and L.A. Wolsey, "Relaxation methods for pure and mixed integer programming problems", *Management Science* 18 (1972) 229–239.
[14] M. Held and R.M. Karp, "A dynamic programming approach to sequencing problems", *SIAM Journal* 10 (2) (1962).
[15] M. Held and R.M. Karp, "The travelling-salesman problem and minimum spanning trees: Part II", *Mathematical Programming*, 1 (1) (1971) 6–25.
[16] M. Held, P. Wolfe and H.P. Crowder, "Validation of subgradient optimization", *Mathematical Programming* 6 (1) (1974) 62–88.
[17] W.W. Hogan, R.E. Marsten and J.W. Blankenship, "The BOXSTEP method for large scale optimization", *Operations Research* 23 (3) (1975) 389–405.
[18] E.L. Lawler, "On scheduling problems with deferral costs", *Management Science* 9 (4) (1963).
[19] T.L. Morin and R.E. Marsten, "Branch and bound strategies for dynamic programming", *Operations Research* 24 (4) (1976) 611–627.
[20] G.L. Nemhauser, *Introduction to dynamic programming* (Wiley, New York, 1966).
[21] A.H.G. Rinnooy Kan, B.J. Lageweg and J.K. Lenstra, "Minimizing total cost in one-machine scheduling", *Operations Research* 23 (5) (1975) 908–927.
[22] A.H.G. Rinnooy Kan, B.J. Lageweg and J.K. Lenstra, private communication (September 1975).
[23] J. Shwimer, "On the n-job, one machine, sequence-independent scheduling problem with tardiness penalties: a branch-bound solution", *Management Science* 18 (6) (1972) 301–313.
[24] V. Srinivasan, "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness", *Naval Research Logistics Quarterly* 18 (3) (1971) 317–327.