Single Machine Scheduling with Flow Time and Earliness Penalties

JONATHAN F. BARD[1], KRISHNAMURTHI VENKATRAMAN[2], and THOMAS A. FEO[3]

[1]*Professor, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, University of Texas, Austin, TX 78712-1063, U.S.A.*; [2]*Graduate Student, Department of Industrial Engineering and Engineering Management, Stanford University, Stanford, CA 94350, U.S.A.*; [3]*Associate Professor, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, University of Texas, Austin, TX 78712–1063, U.S.A.*

Received: 26 February 1992: accepted: 28 July 1992)

Abstract. This paper considers the problem of scheduling n jobs on a single machine to minimize the total cost incurred by their respective flow time and earliness penalties. It is assumed that each job has a due date that must be met, and that preemptions are not allowed. The problem is formulated as a dynamic program (DP) and solved with a reaching algorithm that exploits a series of dominance properties and efficiently generated bounds. A major factor underlying the effectiveness of the approach is the use of a greedy randomized adaptive search procedure (GRASP) to construct high quality feasible solutions. These solutions serve as upper bounds on the optimum, and permit a predominant portion of the state space to be fathomed during the DP recursion.

To evaluate the performance of the algorithm, an experimental design involving over 240 randomly generated problems was followed. The test results indicate that problems with up to 30 jobs can be readily solved on a microcomputer in less than 12 minutes. This represents a significant improvement over previously reported results for both dynamic programming and mixed integer linear programming approaches.

Key words. Single machine scheduling, dynamic programming, greedy heuristics, bicriteria optimization, branch and bound.

1. Introduction

The extensive interest in single machine scheduling (SMS) over the last 30 years stems partly from the fact that effective shop floor control often means developing a schedule that reduces the bottlenecks. In most cases, the primary bottleneck can be found at a single machine or work center. The purpose of this paper is to present a methodology that extends the size of single machine scheduling problems that can be solved to optimality. In particular, for a given set of n jobs with arbitrary due dates, we consider the problem of finding a sequence that minimizes the sum of the corresponding flow time and earliness penalties. In the model, it is assumed that each job is available simultaneously at time zero, and that once started, must be processed without interruption. Further assumptions are that all due dates must be met, and that setup times are sequence independent, implying that they can be added to the processing times.

Journal of Global Optimization **3**: 289–309, 1993.

The flow time penalty is commonly included in scheduling models, and provides a weighted measure of how long it takes to complete a work order. The corresponding function may be viewed as a surrogate for the capital expenses associated with building and operating a facility. When all jobs are weighted equally, it is equivalent to the makespan.

The earliness penalty is used to model the inventory costs that arise at the various stages of production. In fact, a cornerstone of the just-in-time manufacturing philosophy is that excess inventory at any stage is wasteful and should be rigorously avoided. Because value is added to a job as it is processed, the resulting holding costs increase proportionally. The earliness penalty thus represents the cost incurred in storing an item until it is shipped. Here, warehousing must be taken into consideration. If inventories are small and fast moving, storage may be assigned to special areas near the production facility. Large finished goods inventories, however, may occasion additional overhead, rental, and transportation costs.

A restricted version of this problem was first investigated by Fry and Leong (1987) using a mixed integer linear programming (MILP) formulation. They limited their analysis to the case where the flow time penalties are the same and the earliness penalties are the same for all the jobs; no solutions were obtained for problems with $n > 14$. Related work is reported by Faaland and Schmitt (1987) who are primarily concerned with improving existing bounds and developing heuristics, and Sen *et al.* (1988) who derive new bounds for an an enumerative algorithm.

In a review of the SMS literature, Gupta and Kyparisis (1987) offer two different categorizations of the general problem. The first centers on efforts to determine batch sizes when a series of products are to be manufactured on a given machine. The goal here is to strike a balance between setup and inventory holding costs. Notable contributions in this area have been made by Bomberger (1966), and Dobson *et al.* (1987), just to name a few. The latter studied the effect of batching on the total flow time, and provide a variety of heuristics and bounds for the multiproduct case.

The second category deals with fixed lot sizes, and includes problems where the set of jobs must be scheduled to meed one of several objectives, such as minimizing the total (weighted) completion time, minimizing the total (weighted) tardiness, or minimizing the number of late jobs (e.g. see Baker and Bertrand 1981, Bratley *et al.* 1971, Harriri and Potts 1983, and Kanet 1981). The degree of difficulty of the underlying problem depends on the nature of the organizational and technological constraints that are imposed.

The problem considered in this study falls in the second category. In the analysis, we use a branch and bound procedure in the context of dynamic programming (DP) to implicitly generate and explore the state space (Bansal 1980, Morin and Marsten 1977, Potts and Van Wassenhove 1985). At each iteration of the DP, lower bounds are computed by permitting jobs to be split.

Tight upper bounds are obtained at the outset with a randomized adaptive greedy heuristic which provides the key to effective fathoming. Computational results confirm that the presence of good feasible solutions dramatically cuts down the size of the state space. This allows us to readily solve 30 job problems on a microcomputer relying on only 550 kilobytes of memory.

In the next section, notation is defined and the dynamic program is developed. This is followed by the presentation of the lower bounding technique in §3, and a discussion of the greedy heuristic. A procedure for optimally inserting slack time into a given sequence is also detailed. Computational experience is highlighted in §4 along with some insights gained from testing. We close with an assessment of the methodology.

2. Notation and Model Development

The problem addressed consists of a set of independent jobs $J = \{1, \ldots, n\}$ that are to be processed without preemption on a single machine. The jth job is characterized by the following data:

α_j: earliness penalty of job j
λ_j: flow time penalty of job j
p_j: processing time of job j
d_j: due date for job j
c_j: completion time of job j

where all values are assumed to be nonnegative, and p_j integer. Because it is stipulated that all jobs are available at time zero, the flow time and completion time of a particular job are identical. The objective is to find a feasible schedule, σ_f, for J that minimizes $\Sigma_{j=1}^{n} [\lambda_j c_j + \alpha_j(d_j - c_j)]$. For the simple case where all the due dates are equal and $\lambda_j \geq \alpha_j$ for all j, the problem is equivalent to minimizing the weighted sum of completion times, and can be solved by sequencing the jobs in the nondecreasing order of the ratio $p_j/(\lambda_j - \alpha_j)$. Alternatively, if $\alpha_j \geq \lambda_j$ for all j, the problem is equivalent to maximizing the sum of weighted earliness (Chand and Schneeberger 1988).

In general, a schedule may be specified by assigning a start time to each $j \in J$, or, for our purposes, by specifying a feasible sequence, Ω_f, and slack between jobs. Notationally, this is represented by: $\sigma_f = (\langle j_1, s_1 \rangle, \ldots, \langle j_n, s_n \rangle)$, where j_k is the job scheduled in the kth position, and s_k is the amount of slack inserted before it. For every σ_f there is a corresponding flow time f which is equal to the completion time of j_n.

Following the approach of Held and Karp (1962), define a subset of jobs $S = \{j_1, j_2, \ldots, j_i\}$ of cardinality i. Similarly, define a feasible sequence $\Omega_f \leftarrow S$ as a list of jobs that can be processed in the prescribed order without violating the due date constraints. The pair (S, f) corresponds to the *state* of the dynamic

program, while the *stage* is associated with the number of jobs currently under consideration.

The optimal value function, $F_i(S, f)$, or cost-to-go from stage 0 and initial state $(\varnothing, 0)$, to stage i and state (S, f), can be derived using the logic of forward dynamic programming. Assume that the costs-to-go from $(\varnothing, 0)$ to all states at stage i-1 are known, and let job k be the next job to be added to the schedule. At stage i we must consider all states (S', f'), such that S' is any permutation of the set of i jobs containing job k and f' is the flow time for a particular S'. Here, f' ranges between $\Sigma_{j\in S'} p_j$ and $\max[d_k : k \in S']$. The minimum cost-to-go can be computed from the known costs-to-go to stage i–1 by considering all the immediate predecessors of (S', f') at stage i–1 that do not contain job k. The optimal path to (S', f'), must pass through one of these predecessors. Therefore,

$$F_i(S', f') = \alpha_k r(k) + \lambda_k f' + \min_{t\in T}[F_{i-1}(S'\backslash\{k\}, f' - p_k - t)], \tag{1}$$

where $T = \{0, \ldots, f' - \Sigma_{j\in S'\backslash\{k\}} p_j\}$, and $r(k)$ is an earliness function defined as follows:

$$r(k) = \begin{cases} d_k - f' & \text{if } d_k - f' \geqslant 0, \\ \infty & \text{otherwise.} \end{cases}$$

The earliness function in (1) returns the slack time that remains for the candidate job, and is used to ensure that the due dates are met. This is accomplished by setting the slack time to a very large value (∞) whenever the due date constraint of that job is violated. The bounding procedure, described presently, then fathoms the state.

The second term in (1), $\lambda_k f'$, is the flow time penalty for job k that results when it finishes at time f'. These first two terms represent the marginal cost of going from stage i–1 to stage i. The last term is the optimal cost of scheduling the remaining i–1 jobs for a reduced flow time of $f' - p_k - t$, where t is the slack that is inserted before the start of job k.

By extending the analysis so that any unscheduled job may be considered for the ith position, we get the general recurrence relation:

$$F_i(S, f) = \min_{k\in S}[\alpha_k r(k) + \lambda_k f + \min_{t\in T}\{F_{i-1}(S\backslash\{k\}, f - p_k - t)\}], \tag{2}$$

where $i = 1, \ldots, n$; $f = \Sigma_{j\in S} p_j, \ldots, \max_{j\in S}[d_j]$, S ranges over all subsets of size i, and now $T = \{0, \ldots, f - \Sigma_{j\in S} p_j\}$.

The *boundary condition* at stage 0 for the initial state $(\varnothing, 0)$ is $F_0(\varnothing, 0) = 0$. Finally, at stage n, all jobs have been scheduled so it is possible to determine the minimum cost solution by solving

$$\min\left[F_n(J, f) : f \in \left\{\sum_{j=1}^{n} p_j, \ldots, \max(d_j : j = 1, \ldots, n)\right\}\right] \tag{3}$$

and then backtracking to find the corresponding sequence, Ω^*, and slacks, s^*. The range on f ensures that all feasible states at stage n are considered. A small example is presented the Appendix to highlight the computations.

For a reference that problem (2) is *NP-complete*, see the subsection in Garey and Johnson (1979) on "Sequencing to Minimize Weighted Completion Time" on one processor. Informally this result is supported by the observation that (2) is equivalent to a modified TSP in which $m(\geq n)$ cities are partitioned into n disjoint subsets with the objective of finding a minimum cost tour that visits exactly one city in each subset. Dreyfus and Law (1976) show that the DP formulation of this problem has complexity $O(\mu n^2 2^n)$, where μ is the maximum number of cities in any one subset.

3. Solution Methodology

Problem (2) was derived using the *pulling* or recursive fixing logic of dynamic programming where one determines the best (least cost) way to enter a given state from all previous states. Either forward or backward recursion may be used to perform the calculations. In solving problem (2), however, we adopt a *reaching* algorithm which is more efficient when branch and bound is employed. Reaching is strictly a forward scheme where the total cost of going from the origin via a given state to an immediate successor state is computed and compared with the cost of the current path to that state. If the new path offers an improvement, it replaces the incumbent. Denardo and Fox (1979) demonstrate that reaching runs faster and requires less storage than pulling when any of the intermediate states can be fathomed.

Before presenting the algorithm, we introduce some additional notation and discuss the concepts underlying the bounding scheme. First, a distinction must be made between bracketed and unbracketed subscripts. The former is a pointer to the job occupying that position in the schedule, while the latter refers to the job itself. For instance, p_5 is the processing time of job 5, and $p_{(5)}$ refers to the processing time of the fifth job in the schedule. With this in mind, let $\Lambda_{(S,f)}$ be the set of successors of the state (S, f) at stage i, and let ω_k be the window of feasibility for job k. A successor of stage (S, f) is any state $(\hat{S}, \hat{f})$ such that

$$\hat{S} = S \cup \{k\},\ k \in J \backslash S;\ \hat{f} \in [f + p_k,\ f + p_k + \omega_k], \quad \text{and}$$

$$\omega_k = \min[d_k - c_k,\ \min_{j \in \hat{J}} (d_{(j)} - c_{(j)}] \text{ where the inner minimization is}$$

taken over

$$\hat{J} = \left\{ j:\ j = i+2, \ldots, n \text{ such that } d_{(j)} \leq d_{(j+1)} \right.$$

$$\left. \text{and } c_{(j)} = f + p_k + \sum_{m=i+2}^{j} p_{(m)} \right\}.$$

The set $\hat{S}$ is formed by adding one of the remaining (unscheduled) jobs k to S. It is tentatively scheduled as early as possible. The flow time $\hat{f}$ can range over the limits established by ω_k. To test the feasibility of the resulting state, the remaining jobs are tentatively ordered according to their due dates (earliest due date (EDD) first), with no intervening idle times between them. This gives rise to the set $\hat{J}$. The differences between the due date of each of these jobs and its completion time is calculated; ω_k is the minimum value obtained over all the remaining jobs. If ω_k is negative there are no successor states of (S, f) of the form $(S \cup \{k\}, f^k)$, $f^k \geq f + p_k$ since the due date of at least one of the remaining jobs will be violated by any schedule passing through $(S \cup \{k\}, f^k)$. This is based on the following well known result.

PROPOSITION 1. *If a schedule formed by arranging a set of jobs in EDD order with no intervening idle time is infeasible, then no feasible schedule of the jobs exist regardless of the permutation chosen.*

An important feature of the reaching method is that when a stage is entered, the optimal paths and their costs are known for all previous states. The significance of this comes into play in the computation of lower bounds and in the implementation of fathoming procedures. Techniques to reduce the state space are usually based on two approaches. The first is to try to fathom a state by comparing the cost of a (possibly infeasible) schedule through that state with the cost of the best feasible schedule known. The second tries to establish dominance properties which permit the elimination of certain states and all schedules passing through them (e.g., see Erschler *et al.* 1982).

3.1. BRANCH AND BOUND

The idea of using branch and bound to reduce the state space of a dynamic program was first proposed by Morin and Marsten (1976) who applied it to the travelling salesman problem and a nonlinear knapsack problem. Barnes and Vanston (1981) also used it to solve a machine scheduling problem with delay penalties for late jobs, and sequence dependent setup costs and times. They report results for 20 job problems.

To see how a state can be fathomed, assume that cost-to-go to (S, f) is known, but the best schedule for the remaining $n - i$ jobs starting at flow time f is not known. If a relaxation of the latter problem can be solved, then the sum of the cost-to-go to (S, f) and the solution of the relaxed problem will give a lower bound on the cost of the best schedule through (S, f). Notationally, let $R(S, f)$ be a relaxed solution for scheduling the remaining $n - i$ jobs, and $G(S, f)$ be the cost of the best schedule passing through the state (S, f). Then

$$F_i(S, f) + R(S, f) \leq G(S, f). \tag{4}$$

Now, suppose that we have a feasible schedule σ with corresponding cost T_σ. If

$T_\sigma < F_i(S, f) + R(S, f)$ then from (4) we have

$$T_\sigma < G(S, f)\,. \tag{5}$$

Thus, if the cost of the feasible solution is less than the cost of the optimal solution passing through (S, f), all schedules passing through this state can be eliminated from further consideration.

Equation (4) indicates that the tighter the lower bound on the optimal solution, the greater the likelihood of fathoming a state. Similarly, (5) implies that the closer the feasible solution is to the optimum, the greater the number of redundant paths.

3.2. LOWER BOUNDS

Because the relaxed problem must be solved for each state of the DP, the accompanying computational effort should be minimal. At the same time, the solution must provide a close approximation to the best cost of a feasible schedule through the state. These requirements are often in opposition, and hence must be resolved satisfactorily if the bounding scheme is to be effective.

The method developed for generating lower bounds is based in part on the work of Posner (1985) who addresses the problem of minimizing the weighted completion time of n jobs subject to deadlines. Lower bounds are obtained by job splitting. His procedure, denoted as **LB**, is adapted to our problem by partitioning the remaining jobs into two mutually exclusive sets A and B, where

A: the set of remaining jobs whose flow time penalties are less than or equal to their earliness penalties ($\lambda_j \leqslant \alpha_j$).
B: the set of remaining jobs whose flow time penalties are greater than their earliness penalties ($\lambda_j > \alpha_j$);

The following algorithm generates a lower bound on the cost of assigning start times to the jobs in A.

PROCEDURE LOWER_BOUND
Input: α_j, λ_j, p_j, d_j for all $j \in J$, A
Output: Lower bound (c_{assign}) and schedule for jobs in A

Step 0: $c_{\text{split}} = 0$, $d_{\max} = \max[d_j : j \in A]$, $E = \{j : d_j = \max[d_k : k \in A]\}$,

$\bar{E} = A \backslash E$, $w_j = \alpha_j - \lambda_j$, $j \in A$, $c_{\text{initial}} = \sum_{j \in A} \lambda_j d_j$

Step 1: ***If*** $E = \varnothing$ **Then** $c_{\text{assign}} = c_{\text{initial}} + c_{\text{split}}$, **Stop**
Else

$$w_t/p_t = \max[w_j/p_j : j \in E]$$

Endif

Step 2: Let $T = \{k \in \bar{E} : d_k \geq d_{\max} - p_t\}$

If there exists a $k \in T$ such that $\frac{w_k}{p_k} > \frac{w_t}{p_t}$ **Then**

$r \in \{j : d_j = \max[d_k : k \in T \text{ and } w_k/p_k > w_t/p_t]\}$

Go To *Step 3*

Else

Assign job t to start as late as possible (i.e., at time $d_{\max} - p_t$)

$E \leftarrow E \cup T \backslash \{t\}$

$\bar{E} \leftarrow \bar{E} \backslash T$

$d_{\max} = d_{\max} - p_t$

If $E = \varnothing$ **Then**

$E = \{j : d_j = \max[d_k : k \in \bar{E}]\}$

$d_{\max} = \max[d_j : j \in E]$

$\bar{E} \leftarrow \bar{E} \backslash E$

Go To *Step 1*

Endif

Endif

Step 3: Split job t into two jobs, the first of size $d_{\max} - d_r$ and the second of size $p_t - (d_{\max} - d_r)$. Assign weights $w_t(d_{\max} - d_r)/p_t$ and $w_t(p_t - (d_{\max} - d_r))/p_t$ to the two jobs, respectively. Schedule the first job as late as possible (to start at time d_r), and **Set**

$E \leftarrow E \cup \{r\}$, $\bar{E} = \bar{E} \backslash \{r\}$

$c_{\text{split}} = c_{\text{split}} - w_t(d_{\max} - d_r)(p_t - (d_{\max} - d_r))/p_t$

Label the second job t

Set $d_{\max} \leftarrow d_r$, $t \leftarrow r$

Go To *Step 2*

In Step 3, the cost incurred in splitting jobs, c_{split}, must be subtracted from the cost of the assignment in order to ensure a valid lower bound. This is in contrast to **LB** where c_{split}(CBRK) is added to the cost of the assignment to obtain the final cost of the lower bound.

The procedure **LB** can also be applied to the set B with the following modifications:

(i) $d_{\max}$ is initialized to $f + \Sigma_{j \in B}\, p_j$.

(ii) The initial cost of the assignment (c_{initial}) is set to $\Sigma_{j \in B}\, (\lambda_j f + \alpha_j(d_j - f))$. This change is required since the lower bound is computed for an intermediate state at which a partial schedule already exists.

(iii) The due dates used in **LB** are modified to reflect the fact that a flow time of f has already elapsed. This is done by using $d_j - f$ as the effective due date for job j.

(iv) The weights w_j used are set equal to $\lambda_j - \alpha_j$.

(v) In Step 2 of **LB**, job r is identified as follows: $r \in \{j: d_j = \max[d_k: k \in T \text{ and } w_k/p_k > w_t/p_t]\}$. This change is necessary because the assumption that the jobs are indexed in increasing order of their due dates no longer holds.

3.3. DOMINANCE PROPERTIES

If a certain ordering of two jobs in a feasible sequence can be shown to produce a schedule that is never worse than the best possible schedule formed from the sequence created by exchanging these jobs, then the second sequence can be fathomed. A similar statement can be made with regard to the insertion of slack time between two consecutive jobs j and k, whose earliness penalties are greater than their flow time penalties. If job j is not at its due date, the schedule created by increasing its completion time by as much as a single unit, will result in a cost reduction. Thus, the second state $(S, f+1)$ dominates the first (S, f).

More formally, assume the DP is at (S, f), and let job j be the last one scheduled. Define $J' \equiv J \backslash S$, and the parameter $\beta_j \equiv \alpha_j - \lambda_j$ as the difference between the earliness and flow time penalties of job j. The set of states that can be reached from (S, f) are of the form $(S \cup \{k\}, f + p_k + s_k)$, $k \in J'$. The following dominance properties play a significant role in reducing the cardinality of this set.

CASE I – $\beta_j > 0$ (earliness penalty greater than flow time penalty)

(i) $\beta_k > 0$: If $f + p_k < \min[d_j, d_k]$ then only state $(S \cup \{k\}, f + p_k)$ need be reached from (S, f). All other states (with $s_k > 0$) can be fathomed.

Proof: Suppose that state $(S, f + p_k + s_k)$ is reached from the present state. Then if $f + s_k \leq d_j$, the completion time of job j can be moved forward by s_k, resulting in a decrease in cost of $s_k B_j$. If $f + s_k > d_j$, the completion time of job j can be moved up to d_j, resulting in a decrease in cost of $\beta_j(d_j - f)$. Thus, all states with $s_k > 0$ are dominated.

(ii) $\beta_k < 0$: If $f < \min[d_j, d_k]$ then all feasible successors $(S \cup \{k\}, f + p_k + s_k)$ must be considered.

(iii) $\beta_k \leq 0$: If $f + p_k \leq \min[d_j, d_k]$ then no successors of the present state $(S, f + p_k + s_k)$ need be considered.

Proof: Suppose that state $(S, f + p_k + s_k)$ is reached from the present state. If jobs j and k are now interchanged, a cost reduction of $-\beta_k(s_k + p_j) + \beta_j(s_k + p_k)$ results. This follows directly from the fact that $\beta_k \leq 0$ and $\beta_j > 0$. This reduction can be achieved without

violating the feasibility of the remaining jobs since the interchange of the two jobs does not affect the start times of any of the other jobs.

(iv) $\beta_k \leq 0$: If $f + p_k > \min[d_j, d_k]$ then only state $(S, f + p_k)$ need be considered.

Proof: The state (S, f) is feasible and therefore d_j is greater then or equal to the flow time, f. If $d_j < f + p_k$, the jobs j and k cannot be exchanged without violating the feasibility of j. As a result, the state $(S \cup \{k\}, f + p_k)$ must be considered. On the other hand, any state $(S \cup \{k\}, f + p_k + s_k)$ with $s_k > 0$, will be dominated by $(S \cup \{k\}, f + p_k)$ due to the associated reduction in cost $-\beta_k s_k$. If d_k is less than $f + p_k$, the state $(S \cup \{k\}, f + p_k)$ is infeasible.

CASE II – $\beta_j \leq 0$ (flow time penalty greater than or equal to earliness penalty)

(i) $\beta_k > 0$: All states $(S \cup \{k\}, f + p_k + s_k)$ must be considered.

(ii) $\beta_k \leq 0$: Only the state $(S \cup \{k\}, f + p_k)$ need be considered.

Proof: Because β_j and β_k are both less than or equal to zero, it can never be optimal to insert idle time between jobs j and k.

3.4. FINDING FEASIBLE SOLUTIONS

In our experience, the most promising heuristics for generating feasible solutions to difficult combinatorial optimization problems incorporate some amount of randomness in the search (e.g., see Bard and Feo 1989). Many of the recent approaches derive from analogs in genetic and physical systems. For example, simulated annealing is a technique which is based on the probabilistic behavior of atoms and molecules in a thermal system being cooled in a controlled environment (Kirkpatrick *et al.* 1983). At each step in the algorithm, a set of "good" choices is available, but the "best" isn't always chosen. The motivation for this strategy is to avoid getting trapped at a local solution. By making a "less than optimal" decision with a certain probability, the same solution doesn't always arise. Thus, many different paths can be explored through repeated trials.

In this study, we develop a greedy randomized adaptive search procedure (GRASP) for heuristically solving the SMS problem (2) that exploits these ideas. GRASP combines the power of greedy selection rules, randomization, and conventional search techniques such as neighborhood exchanges. The approach is iterative with each GRASP iteration consisting of a construction phase and a local minimization phase. More specifically, suppose there is a set of elements whose costs depend upon their position in a sequence. At every stage in the construction phase, one of the remaining elements is selected and added to the partial sequence. A greedy function is used to estimate the cost of the elements that can be included at that stage. Two important aspects of the heuristic come into play here. First, the computation of these costs is restricted to depend only on the set

of remaining elements. This is the adaptive aspect of the method. Second, a greedy rule is used to compute the costs of the remaining elements and to determine their suitability relative to the others. The "better" elements are those with the smaller costs.

A restricted candidate list of elements is compiled next and one of the elements (jobs) is chosen. A purely greedy approach would pick the element with the least cost. This is analogous to steepest descent. While such a strategy might produce good results, it is unlikely to yield optimal solutions with any degree of consistency. If the list is perturbed, however, there is a greater chance of obtaining an improved solution over many GRASP iterations. This is the motivation for the randomization of the procedure. Instead of picking the best element, a probability measure is imposed on the candidates and one is picked accordingly. In practice, an upper limit is placed on the size of the list to restrict the selection to those that have relatively favorable costs. This prevents the greedy function from being overly compromised. One approach for doing this is to include all elements whose costs are within a fixed percentage of the cost of the best element. The choice of the list restriction method is an art that requires some insight into the particular problem being solved. The size should be large enough so that many different solutions are examined, but should not be too large to allow degradation of the quality of the solutions obtained.

For problem (2), the earliness and flow time penalties of the jobs as well as their due dates are used to estimate the cost of assigning a job to the next position in the sequence. This is done in two stages; all jobs in J' are considered. To begin, a prospective job, call it $j \in J'$, is selected and assigned a completion time so that none of the remaining jobs in $J' \backslash \{j\}$ violate their due date constraints. (Feasibility is checked by constructing the EDD sequence and appealing to Proposition 1. The procedure for making the assignment is, in part, based on ω_j, α_j, and λ_j; see Feo *et al.* 1991 for a detailed explanation). The cost incurred by scheduling job j to finish by this time is calculated. Next, completion times are assigned to each of the remaining jobs based on their due dates and the completion time of j. Costs corresponding to these jobs are computed and the total cost of assigning j to the next position in the sequence is found by adding these together. This is done for all $j \in J'$ that can be feasibly scheduled at the current position. A restricted candidate list is then constructed and a job randomly selected.

Once a feasible schedule is generated, two additional procedures are applied to assure that a local optimum has been obtained. The first, INSERT_SLACK, efficiently inserts optimal slack times between jobs once a sequence is specified. The details are given in the next section. The second, POST_PROCESSOR, performs pairwise swaps and then calls INSERT_SLACK to reoptimize. When no further improvement is possible with respect to the pairwise swaps, the incumbent is updated and the next GRASP iteration is executed.

The general flow of the GRASP is outlined below. The details on how the final

set of parameters was chosen, and how well the heuristic performs over a wide range of problems are discussed in Feo *et al.* (1991). In the actual implementation, the candidate list is restricted to three jobs (clim). Many other restrictions were investigated but proved to be inferior for this SMS problem.

In the development, we make use of the following additional notation:

clim:	maximum length of candidate list
limit:	number of GRASP iterations
θ_j:	marginal slack time of job j
Γ:	candidate list
η:	integer index
ρ_j:	sum of processing times through the jth job in the current schedule
ω_j:	feasible delay window of jth job in current schedule

```
PROCEDURE GRASP_OUTLINE
ρ0 = 0 (initialize sum of processing times)
Construct EDD list
Do k = 1, limit (GRASP loop)
  J' ← J (initialize set of unscheduled jobs)
  Do i = 1, n (construction loop)
    Do j ∈ J' (greedy rule loop)
      Compute ωj for job j
      Select completion time of job j
      Estimate completion time of all other jobs in J'
      Compute cost of job j
    Next j
    Find minimum cost job jmin
    Γ ← {jmin} (initialize candidate list)
    While [(|Γ| ≤ clim) and (J'\Γ ≠ ∅)] (construct candidate list)
      Find next smallest minimum cost job jmin ∈ J'\Γ
      Γ ← Γ ∪ {jmin}
    Endwhile
    η = UNIFORM[1, |Γ|] (select an integer at random)
    Add jη to sequence
    J' ← J'\{jη}
    ρi = ρi−1 + ρη (update sum of processing times)
  Next i
  Call INSERT_SLACK (Call optimal slack time insertion routine)
  Update incumbent schedule
  Call POST_PROCESSOR (Call neighbor exchange routine)
Next k
```

3.5. OPTIMAL IDLE TIME INSERTION

The performance of the GRASP is greatly enhanced by the fact that it is possible to obtain an optimal schedule for a given sequence with very little effort. The proposed method uses a marginal cost, θ_j, as the criterion to determine whether or not idle time should be inserted before job $j \in J$. For a given sequence, an initial schedule is constructed without any slack. Each job (j) has a window of feasibility, ω_j, associated with it that determines how much idle time, s_j, can be feasibly inserted. Starting with the last job (n), the completion time of each job (j) is moved up to the limit imposed by ω_j whenever $\theta_j < 0$. The completion times of the other jobs are adjusted accordingly.

The marginal cost for a particular job is calculated by maintaining information about the completion times of the jobs that follow it. This value measures the increase or decrease in the cost of the schedule that results from increasing the completion times of these jobs by one unit. If θ_j is negative, it indicates that the cost of the schedule will decrease by this amount for every unit of time job j is delayed. The maximum idle time that can be inserted is governed by the window of feasibility.

PROCEDURE INSERT_SLACK
Input: α_j, λ_j, p_j, ρ_j for all $j \in J$
Output: c_j, s_j for all $j \in J$

$\omega_n = d_{(n)} - \rho_n$ (compute feasible delay window for last job)
$\theta_{(n)} = \lambda_{(n)} - \alpha_{(n)}$ (compute marginal slack time cost for last job)
Do $j = n, 1$
  If $(\theta_{(j)} > 0)$ **Then** (test marginal slack time cost of jth job)
    $s_j = 0$ (set slack time of jth job to zero)
    $\theta_{(j-1)} = \theta_{(j)} + \lambda_{(j-1)} - \alpha_{(j-1)}$ (compute marginal cost of next job)
  Else
    $s_j = \omega_j$ (set slack time of jth job to its feasible delay window)
    $\theta_{(j-1)} = \lambda_{(j-1)} - \alpha_{(j-1)}$ (compute marginal cost of next job)
  Endif
  $\omega_{j-1} = \text{MIN}[\omega_j, d_{(j-1)} - \rho_{j-1}]$ (compute feasible delay window of next job)
Next j
$c_{(0)} = 0$ (initialize earliest start time of "dummy" job)
Do $j = 1, n$ (set new schedule)
  $c_{(j)} = \text{MAX}[c_{(j-1)} - p_{(j-1)}, \rho_{j-1} - s_{j-1}]$ (compute completion time of jth job)
  $s_j = c_{(j)} - (c_{j-1}) - p_{(j-1)}$ (compute slack time of jth job)
Next j

PROPOSITION 2. *The procedure INSERT_SLACK produces an optimal schedule for a given sequence.*

The optimality of the procedure can be inferred from the following observations. At each iteration the decision to change the completion time of the current job depends on its marginal cost. This value depends on the penalties of the job itself and those of the jobs succeeding it. Slack is inserted before job (j) only if $\lambda_{(j)} - \alpha_{(j)} < 0$ is sufficiently negative to offset the cost of delaying the jobs that have already been scheduled. If this is the case, job (j) is moved as far forward as possible and its marginal cost is set to 0. If the marginal cost of (j) is positive, no action is taken and its immediate predecessor is considered. Finally, because each job is initially scheduled at its earliest possible start time, feasibility is always maintained.

3.6. COMPUTATIONAL COMPLEXITY

In the initialization phase of the algorithm, the EDD list is constructed. This is an $O(n \log n)$ operation. The selection of a job for a position in the schedule uses a candidate list which is ordered by cost. The cost for a job is calculated taking into account its effect on the remaining jobs. This can be done in $O(n)$ time, implying that the construction of the candidate list is an $O(n^2)$ operation. Because the list must be constructed once for each position in the schedule, the order of the basic heuristic is $O(n^3)$. With regard to post processing, note that a swap requires three operations. The first is the removal of idle times from the schedule. This is an $O(n)$ operation. Next, a check for feasibility of the new schedule must be made. This can be performed in constant time or equivalently $O(1)$. Finally, the insertion of idle time in the new schedule is an $O(n)$ operation. Therefore, each swap is $O(n)$. Assuming that κ swaps are made at each iteration, the procedure runs in $O(\kappa n)$ time.

4. Computational Experience

In order to evaluate the performance of the proposed methodology, a series of data sets for 15, 20, 25 and 30 job problems was randomly generated and solved. Each data set was subdivided into three categories: Type I – the earliness penalty and the flow time penalty are approximately equal for all jobs; Type II – the earliness penalty is greater than the flow time penalty for about two-thirds of the jobs; and Type III – the earliness penalty is greater than the flow time penalty for about one-third of the jobs.

In addition to this classification, two parameters known as the earliness factor e and the due date range r, were used to characterized each data set (see Sen *et al.* 1988). The following combinations of (e, r) were tested: $(0, 1/4)$, $(0, 3/4)$, $(1/4, 1/2)$, and $(1/2, 1)$. Finally, the values of the parameters α_j, λ_j, and p_j were specified by generating three random integers in the interval [0, 10] and assigning one to each. Due dates for the jobs were selected at random from the range:

$$\left[\sum_{j=1}^{n} p_j(1+e-r), \ldots, \sum_{j=1}^{n} p_j(1+e+r)\right].$$

The experimental design consisted of 5 cases for each of the 4 job sizes, 3 types, and 4 (e, r) pairs, giving a total of 240 instances. In the testing, the GRASP is run first to furnish an upper bound for the DP. The high quality of the solutions obtained at this stage argued against further GRASP runs at intermediate states. The empirical statistics collected included (1) the time taken by the DP to either solve the problem or to exceed available memory, and (2) the number of states visited. A third statistic of interest is the size of the underlying state space. However, this value is not easily computed so an approximation was derived based on $\Sigma_j\, p_j$ and $\max[d_j : j \in J]$. A comparison between the actual size and the number of states visited is a good indication of the efficiency of the dominance rules used to fathom states.

4.1. RESULTS

All codes used in the study were written in *C* by the authors using the Microsoft Compiler. The state data is stored as linked lists, and hashing functions are used to search for previously generated states during the recursion. Testing was done on a PS/2 Model 80 with approximately 550 kilobytes of available memory. Because the state space grows exponentially, this number is of particular importance. The branch and bound code requires that all states reached from the generating state, as well as those at the next stage be kept in RAM. The remainder can be relegated to secondary storage and retrieved if necessary when recovering the optimal solution.

Computational results are reported in Table I by problem size and type. Each

Table I. Results for the DP branch and bound methodology

No. of Jobs	Problem Type	Avg. Time (secs)	Std. Deviation (secs)	Avg. No. of States Visited	Std. Deviation (states)	No. Opt. Soln. GRASP
15	I	7.45	1.96	60.55	57.08	20
15	II	7.93	7.03	113.45	127.84	20
15	III	7.85	2.26	46.80	42.86	20
20	I	18.34	8.39	126.15	81.85	20
20	II	34.45	36.80	312.60	278.59	20
20	III	15.91	8.76	111.75	117.76	20
25	I	58.74	20.89	240.15	182.67	20
25	II	157.92	299.05	1040.15	1494.78	20
25	III	33.65	51.62	618.15	1947.09	19
30	I	368.45	787.20	2934.17	6684.39	20
30	II	643.55	992.93	2661.84	3149.96	20
30	III	200.95	518.68	2195.11	3838.08	19

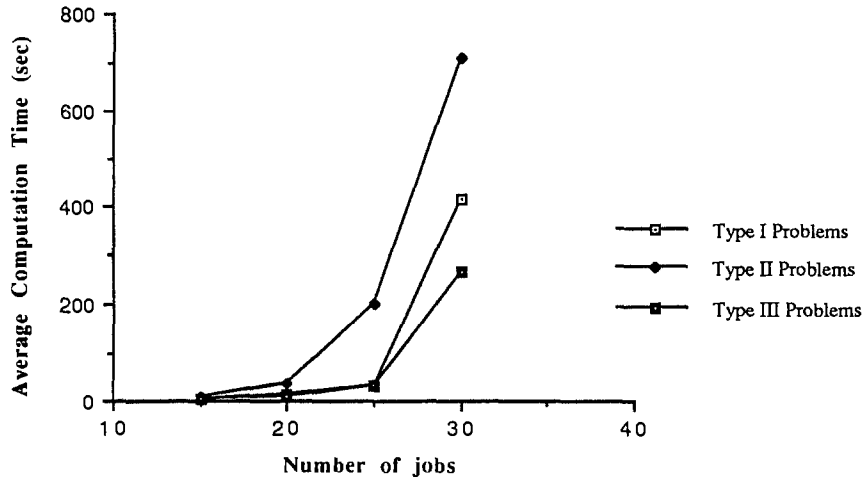

Fig. 1 Relationship between problem size and solution time.

line represents an average of 20 instances, and contains information on the 'average time taken by the DP', the corresponding standard deviation, the 'average number of states visited', the corresponding standard deviation, and the number of times the GRASP found the optimal schedule. As can be seen, there was only one case in the 25 job problem set, and one case in the 30 job problem set where this was not true. In each instance, the GRASP was run for 100 iterations but in all but a handful of cases, found the best solution in five or few tries. CPU time averaged 1 second for the 15 job problems and 10 seconds for the 30 job problems. More details are provided in Feo *et al.* 1991.

The DP solved all instances to optimality, although time and memory requirements grew sharply with n. The graphs in Figures 1 and 2 chart the accompanying performance for the three classes examined. It is interesting to note that the algorithm had the most difficulty with the Type III problems. Recall that jobs in this class are 67% more likely to have earliness penalties greater than their flow time penalties, so the size of the set A used in computing the lower bounds will be on average twice that of B. In light of this ratio, the results imply that the procedure used to find lower bounds is less effective for those jobs wanting to be scheduled as close to their due dates as possible.

4.2. DISCUSSION

In general, we found that the running time of the DP depends greatly on the quality of the upper bound. To cite one example, consider the sixth problem of Type III in the 30 job set whose optimal solution is 27,294. The solution obtained by the GRASP was 27,297. The running time of the DP using 27,297 as a bound is 572 seconds and the number of states visited is 3419. When the problem was resolved using the optimal solution as the upper bound, the running time of the

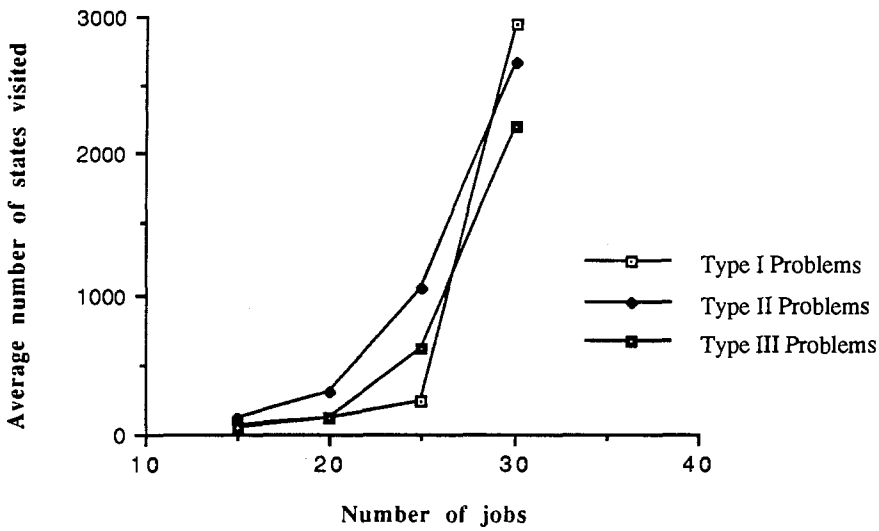

Fig. 2. Relationship between problem size and no. of states visited.

DP decreased to 198 seconds and the number of states visited fell to 1223. The implication is that a very small improvement in the upper bound can result in a dramatic improvement in the performance of the DP.

A second means of evaluating the performance of the branch and bound approach is by comparing the potential size of the unrestricted state space with the number of states actually generated. The former is of order $O(\mu 2^n)$, where $\mu = \max[d_j - p_j : j \in J]$. Table I reveals the average number of states visited by the DP for the 30 job problems is about 3000 in the worst case. This is a minute fraction of the total number that arise if no dominance rules or bounding techniques are employed.

5. Summary and Conclusions

The test results for the DP highlight the effectiveness of the proposed branch and bound methodology in solving this single machine scheduling problem. Optimal solutions to 30 job problems are routinely obtained in less that 12 minutes, regardless of the data characteristics, and with little variation in computational effort. Standard deviations for processing times and states visited are of the same order of magnitude as their means.

In general, by combining efficient heuristics with dominance rules and tight lower bounds, we have been able to consistently solve problems from 50% to over 100% larger than those reported in the literature. One factor contributing to our success has been the GRASP which generates good feasible solutions quickly. And because its running time is polynomial, it can be used on much larger problems. The testing showed that the GRASP found the optimal solution in 238 out of 240 instances.

An interesting extension to the current problem involves the incorporation of sequence dependent setup costs in the model. In this case, a third component corresponding to the last job scheduled would have to be added to the vector defining the state. This would cause the size of state space to expand by a factor of n. Finally, we note that the methodology can be easily extended to accommodate a variety of different terms in the objective functions such as lateness penalties, weighted completion times, and tardiness factors just to name a few.

Appendix

The example presented below demonstrates the mechanics of the reaching method. The data for the problem are contained in Table A1. Three jobs must be scheduled, all of which are available at time zero.

Table A1. Data for example

JOB	p_j	d_j	λ_j	α_j
1	6	10	1	5
2	3	10	2	1
3	3	15	1	4

The algorithm starts at $(\varnothing, 0)$ and reaches out to all successors at stage 1. The states, $\Lambda_{(\varnothing, 0)}$, that can be generated initially are listed in the first column of the following tableau.

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1\}, 6)$	26	26	26	$(\varnothing, 0)$
$(\{1\}, 7)$	22	22	22	$(\varnothing, 0)$
$(\{2\}, 3)$	13	13	13	$(\varnothing, 0)$*
$(\{2\}, 4)$	14	14	14	$(\varnothing, 0)$

* Determined to be on the optimal path during backtracking.

The second column identifies the marginal cost incurred by adding a job to the set associated with the state under investigation. For example, the first state reached from $(\varnothing, 0)$ is $(\{1\}, 6)$. Here, job 1 is added with marginal cost $6\lambda_1 + (10 - 6)\alpha_1 = 6 + (4)(5) = 26$. The cost-to-go is then added to this value giving the total cost of reaching the current state via the generating state (see column 3). If the former was previously generated, its cost-to-go is updated and listed in column 4. The last column contains the corresponding decision (predecessor) of the current state. This value is used in backtracking to recover the optimal path.

At the next iteration, successors of the states at stage 1 are generated. Starting with the first state, $(\{1\}, 6)$, the following states at stage 2 are reached.

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2\},9)$	19	45	45	$(\{1\},6)$
$(\{1,2\},10)$	20	46	46	$(\{1\},6)$

The next state is $(\{1\},7)$ which produces the following tableau.

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2\},10)$	20	42	42	$(\{1\},7)$

Note that the state $(\{1,2\},10)$ is reached from the state $(\{1\},6)$ as well. The total cost-to-go to this state via $(\{1\},7)$ is 46, whereas the cost-to-go via $(\{1\},6)$ is 42. This results in a change in the decision pointing to $(\{1\},7)$ rather than $(\{1\},6)$.

The states reached from the remaining states at stage 1 are listed below, along with other pertinent data.

CURRENT STATE: $(\{2\},3)$

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2\},9)$	14	27	27	$(\{2\},3)$
$(\{1,2\},10)$	10	23	23	$(\{2\},3)$

CURRENT STATE: $(\{2\},4)$

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2\},10)$	10	24	23	$(\{2\},3)$

All states at stage 1 have now been considered. The updated list of states at stage 2 is as follows.

State	Best Cost	Decision
$(\{1,2\},9)$	27	$(\{2\},3)$
$(\{1,2\},10)$	23	$(\{2\},3)^*$

Using the same procedure as before, the states at stage 3 are generated.

CURRENT STATE: $(\{1,2\},9)$

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2,3\},12)$	24	51	51	$(\{1,2\},9)$
$(\{1,2,3\},13)$	21	48	48	$(\{1,2\},9)$
$(\{1,2,3\},14)$	18	45	45	$(\{1,2\},9)$
$(\{1,2,3\},15)$	15	42	42	$(\{1,2\},9)$

CURRENT STATE: $(\{1,2\},10)$

State	Marginal Cost	Total Cost	Best Cost	Decision
$(\{1,2,3\},13)$	21	44	44	$(\{1,2\},10)$
$(\{1,2,3\},14)$	18	41	41	$(\{1,2\},10)$
$(\{1,2,3\},15)$	15	38	38	$(\{1,2\},10)$

The list of the states at stage 3 is given below.

State	Best Cost	Decision
$(\{1,2,3\},12)$	51	$(\{1,2\},9)$
$(\{1,2,3\},13)$	44	$(\{1,2\},10)$
$(\{1,2,3\},14)$	41	$(\{1,2\},10)$
$(\{1,2,3\},15)$	38	$(\{1,2\},10)^*$

At this point, all the jobs have been scheduled so the optimal path can be uncovered. From the last tableau, we see that the schedule with the least cost has a flow time of 15. The decision made at this state is $(\{1,2\},10)$. This implies that job 3 is the last to be scheduled, and finishes at 15. Also, since the completion time of the remaining jobs is bounded by 10, the slack time inserted before job 3 is $c_3 - p_3 - 10 = 15 - 3 - 10 = 2$.

Continuing to backtrack, we see that the predecessor of $(\{1,2\},10)$ is $(\{2\},3)$. Thus, job 1 is scheduled at stage 2 and finishes at 10. The slack time is $10 - 6 - 3 = 1$. The last state that has to be considered is $(\{2,3)$. Job 2 is the only one remaining and is scheduled without slack. The final sequence is $2 \rightarrow 1 \rightarrow 3$ as shown in Table A2.

Table A2. Optimal schedule for example

JOB	c_j	s_j	COST
2	3	0	13
1	10	1	23
3	15	2	38

Acknowledgement

This work was supported by a grant from the Texas Higher Education Coordinating Board's Advanced Research and Technology Programs.

References

Baker, K. R. and Bertrand, J. W. M. (1981), A Comparison of Due Date Selection Rules, *AIIE Transactions* **13**, 123–131.

Bansal, S. P. 1980), Single Machine Scheduling to Minimize Weighted Sum of Completion Times with Secondary Criterion – a Branch and Bound Approach, *European Journal of Operational Research* **5**, 177–181.

Bard, J. F. and Feo, T. A. (1989), Operations Sequencing in Discrete Parts Manufacturing, *Management Science* **35**(2), 249–255.

Barnes, J. W. and Vanston, L. K. (1981), Scheduling Jobs with Linear Delay Penalties and Sequences Dependent Setup Costs, *Operations Research* **29**(1), 146–160.

Bomberger, E. E. (1966), A Dynamic Programming Approach to a Lot Size Scheduling Problem, *Management Science* **20**, 101–109.

Bratley, P., Florian, M., and Robillard, P. (1971), Scheduling with Earliest Start and Due Date Constraints, *Naval Research Logistics Quarterly* **18**, 511–517.

Chand, S. and Schneeberger, H. (1988), Single Machine Scheduling to Minimize Weighted Earliness Subject to No Tardy Jobs, *European Journal of Operational Research* **34**, 221–230.

Denardo, E. V. and Fox, B. L. (1979), Shortest Route Methods; 1. Reaching, Pruning and Buckets, *Operations Research* **27**, 161–186.

Dobson, G., Karmarkar, U. S., and Rummel, J. F. (1987), Batching to Minimize Flow Times on One Machine, *Management Science* **33**, 784–799.

Dreyfus, S. E. and Law, A. (1976), *The Art and Theory of Dynamic Programming*, Academic Press, New York.

Erschler, J., Fontan, G., Merce, C., and Roubellat, F. (1982), Applying New Dominance Concepts to Job Schedule Optimization, *European Journal of Operational Research* **11**, 60–66.

Faaland, B. and Schmitt, T. (1987), Scheduling Tasks with Due Dates in a Fabrication/Assembly Process, *Operations Research* **35**, 378–388.

Feo, T. A., Venkatraman, K., and Bard, J. F. (1991), A GRASP for a Difficult Single Machine Scheduling Problem, *Computers & Operations Research* **18**(8), 635–643.

Fry, T. D. and Leong, G. K. (1987) A Bi-Criterion Approach to Minimizing Inventory Costs on a Single Machine When Early Shipments Are Forbidden, *Computers & Operations Research* **14**, 363–368.

Garey, M. R. and Johnson, D. S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco.

Gupta, S. K. and Kyparisis, J. (1987), Single Machine Scheduling Research, *Omega* **15**, 207–227.

Harriri, A. M. A. and Potts, C. N. (1983), An Algorithm for Single Machine Sequencing with Release Dates to Minimize Total Weighted Completion Time, *Discrete Applied Math* **5**, 99–109.

Held, M. and Karp, R. M. (1962), A Dynamic Programming Approach to Sequencing Problems, *J. SIAM* **10**, 196–210.

Kanet, J. J. (1981), Minimizing the Average Deviation of Job Completion Times about a Common Due Date, *Naval Research Logistical Quarterly* **28**, 643–651.

Kirkpatrick, S., Gelatt, Jr., C. D., and Vecchi, M. P. (May 13, 1983), Optimization by Simulated Annealing, *Science* **220**(4598), 671–680.

Morin, T. L. and Marsten, R. E. (1976), Branch-and-Bound Strategies for Dynamic Programming, *Operations Research* **24**(4), 611–627.

Posner, M. E. (1985), Minimizing Weighted Completion Times with Deadlines, *Operations Research* **33**, 562–574.

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

Sen, T., Raizadeh, F. M. E., and Dileepan, P. (1988), A Branch and Bound Approach the Bicriterion Problem Involving Total Flow Time and Range of Lateness, *Management Science* **34**, 254–260.