Journal of Applied Intelligence 3, 159–172 (1993)
© 1993 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Intelligent Scheduling with Tabu Search: An Application to Jobs with Linear Delay Penalties and Sequence-Dependent Setup Costs and Times

MANUEL LAGUNA
Graduate School of Business and Administration, Campus Box 419, University of Colorado, Boulder, CO 80309-0419
MANUEL@MAYAN.COLORADO.EDU

J. WESLEY BARNES
Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, ETC 5.128D, The University of Texas at Austin, Austin, Texas 78712
BARNES@EMX.UTEXAS.EDU

FRED GLOVER
Graduate School of Business and Administration, Campus Box 419, University of Colorado, Boulder, CO 80309-0419
GLOVER_F@CUBLDR.COLORADO.EDU.

Received December 27, 1990; Revised January 8, 1993

Abstract. In this article we study the *tabu search* (TS) method in an application for solving an important class of scheduling problems. Tabu search is characterized by integrating artificial intelligence and optimization principles, with particular emphasis on exploiting flexible memory structures, to yield a highly effective solution procedure. We first discuss the problem of minimizing the sum of the setup costs and linear delay penalties when N jobs, arriving at time zero, are to be scheduled for sequential processing on a continuously available machine. A prototype TS method is developed for this problem using the common approach of exchanging the position of two jobs to transform one schedule into another. A more powerful method is then developed that employs insert moves in combination with swap moves to search the solution space. This method and the best parameters found for it during the preliminary experimentation with the prototype procedure are used to obtain solutions to a more complex problem that considers setup times in addition to setup costs. In this case, our procedure succeeded in finding optimal solutions to all problems for which these solutions are known and a better solution to a larger problem for which optimizing procedures exceeded a specified time limit (branch and bound) or reached a memory overflow (branch and bound/dynamic programming) before normal termination. These experiments confirm not only the effectiveness but also the robustness of the TS method, in terms of the solution quality obtained with a common set of parameter choices for two related but different problems.

Key words: Production scheduling, tabu search, combinatorial optimization

1. Introduction

In this article, we study an application of *tabu search* (TS) to an important class of machine scheduling problems. Tabu search is founded on integrating problem-solving principles that span the fields of artificial intelligence and mathematical optimization, exploiting flexible memory structures to obtain optimal or near optimal solutions with a high degree of efficiency. We first consider the single-machine scheduling problem with linear delay penalties and setup cost dependencies. At time zero, N jobs simultaneously arrive at a continuously available machine. Each job i (i = 1, 2, . . . , N) requires t_i units of time on the machine, and a penalty p_i is charged for each unit of time that job commencement is delayed after time zero; s_{ij} is the setup cost of scheduling job j immediately after job i. The jobs are indexed according to their *natural order* [1], i.e., $i < j$ implies that $\frac{t_i}{p_i} \leq \frac{t_j}{p_j}$. Two dummy jobs, 0 and N + 1, are included in every schedule, where $t_0 = t_{N+1} = 0$ and $p_0 = p_{N+1} = 0$. The costs $s_{0,j}$ and $S_{i,N+1}$ are considered to be the initial setup cost and the teardown cost, respectively. A *schedule* has the form

$$\Pi = \{0, \pi(1), \pi(2), \ldots, \pi(N), N + 1\}$$

where $\pi(i)$ is the index of the job in position *i* of the schedule. The objective is to minimize the sum of the delay and setup costs for all jobs. In mathematical terms, we desire to

$$\begin{aligned} &\text{Minimize } F(\Pi) = D(\Pi) + S(\Pi) \\ &\text{where} \quad D(\Pi) = \sum_{i=1}^{N} d_{\pi(i)} p_{\pi(i)} \\ &\qquad\quad S(\Pi) = s_{0,\pi(1)} + \sum_{i=1}^{N-1} s_{\pi(i),\pi(i+1)} + s_{\pi(N),N+1} \\ &\text{and} \quad d_{\pi(i)} = \sum_{j=1}^{N-1} t_{\pi(j)},\ i = 2, \ldots, N \text{ and } d_{\pi(1)} = 0 \end{aligned} \qquad \text{(P1)}$$

The available optimal schedule algorithms for P1 are based on branch and bound and dynamic programming [2]. Unfortunately, the explosion of dimensionality and the fact that existing methods require an assignment problem (or assignment problem relaxation) to be solved a very large number of times has limited their power to the solution of small problems. Barnes and Vanston [2] developed three branch and bound algorithms using natural order branching (BB1), minimum bound branching (BB2), and priority branching (BB3). A lower bound obtained by solving the assignment problem was used by the BB algorithms; however, after an attempted solution of a 15-job problem did not terminate before 100 CPU seconds on a CDC 6600, the assignment problem bound was not used in further BB experiments. Morin and Marsten's dynamic programming/branch and bound (DPBB) approach [3] was also applied in [2] to reduce the memory and computational requirements. For the solution of 20-job problems, BB3 was prematurely curtailed, and the curtailed solution was used as the starting point for BB1, BB2, and DPBB. Even then, BB1 and BB2 failed to terminate before the time limit of 100 seconds in all but one of the 20-job problems. The most promising optimal algorithm for P1 developed by Barnes and Vanston is DPBB, which solved all of the 20-job problems in an average time of 42.9 seconds (which includes 10 seconds taken by BB3 to provide the starting solution). However, DPBB was tested with several 25-job problems, and in each of these problems storage limitations were encountered at fairly low stages.

The primary purpose of our current research is to study larger problem instances of P1 and the quality of the solutions obtained for them with the TS method. (For background, see, for example, [4–7].) This method requires minimal computational resources, allowing relatively large problems to be approximately solved on a microcomputer using a general-purpose language. Recent studies have demonstrated that tabu search is able to obtain significantly higher-quality solutions and optimal solutions with substantially greater frequency than alternative approaches in a variety of domains, including graph coloring [7], vehicle routing [9], and quadratic assignment [10] problems. In the area of scheduling, although examining a somewhat different problem than the one addressed here, the study by Widmer and Hertz [11] compares tabu search to six alternative approaches to the m-machine, n-job flow shop problem. The authors find that tabu search obtains better solutions than all other methods in up to 90% of the cases. More recent studies surveyed in [6,7] encouraged us to develop a tabu search method for the solution of P1, with the additional challenge of comparing this approach to a tailored, special-purpose optimization method (which has been reported to be more effective than competing optimization procedures for this problem). A further goal of our research effort is to show the robustness of our method by adapting it for the approximate solution of a related problem that includes *setup times* (problem P2 in section 7).

As described in section 2, the TS method was used to redirect the steps of a simple *hill-climbing* heuristic as a metaprocedure to overcome local optimality. Definitions associated with the TS method and the data structures we used in the present setting are given in section 3. In section 4, a prototype TS method tailored for the solution of P1 is described. Section 5 illustrates our TS procedure by identifying the contents of the tabu status arrays that guide the solution process at each iteration when applied to an example five-job problem. Preliminary computational experiences are reported in section 6. An improved TS method is described in section 7, and computational experiences with the more complex problem P2 are presented. Final remarks are given in section 8.

2. A Hill-Climbing Heuristic

P1 is an optimization problem of the following form. Given jobs $i = 1, \ldots, N$, we seek a permutation, or schedule, that may be described as assigning each i to one of the N possible positions. The vector of positions uniquely identifies the schedule, and the goal is to minimize a function of this vector, i.e., minimize $F(\Pi)$. The preliminary selection of the class of moves to be embedded within the TS method consists of common pairwise exchanges that swap the position of two jobs to transform one schedule into another. Suppose, in a given schedule, job $\pi(i)$ precedes, but is not necessarily adjacent to, job $\pi(j)$. A *move* is a recording of *only* jobs $\pi(i)$ and $\pi(j)$ such that job $\pi(i)$ is moved to position j and job $\pi(j)$ is moved to position i. The *move value* is the difference between the value of the objective function after the move, $F(\bar{\Pi})$, and the value of the objective function before the move, $F(\Pi)$, i.e.,

$$\text{move\_value} = F(\bar{\Pi}) - F(\Pi)$$

For comparative purposes, it is convenient to begin by noting the form of a simple *hill-climbing* heuristic that uses the same swap moves (the "hill" is inverted for minimization). The organization of the heuristic presented in figure 1 has been chosen to allow direct elaboration into a simple TS procedure. Note in particular that a best member of the *candidate moves* was chosen at each step. Often hill-climbing methods are organized to accept any improving move. The more aggressive orientation of choosing the best member is particularly relevant to tabu search after a local optimum has been found. The fact that examination is limited to candidate moves, however, allows the chance to define the set of such moves dynamically. We will superimpose tabu search on this heuristic, with the goal of providing a guiding framework to overcome the strong tendency of the hill-climbing process to become trapped at a local optimum.

3. Definitions

The fundamental process by which tabu search seeks to transcend local optimality is to introduce a mechanism to make certain moves forbidden (tabu). In this context, there are four key elements to consider:

1. To identify one or more *attributes* of a move that will be used to create the tabu classifica-

```
Generate starting solution $\Pi^*$;
Evaluate $F(\Pi^*)$;
do {
    best_move_value ← ∞;
    for (all candidate moves) {
        Evaluate move_value;
        if (move_value < best_move_value) {
            best_move_value ← move_value;
            best_move ← current move;
        }
    }
    if (best_move_value < 0) {
        Execute best_move;
        $F(\Pi^*) \leftarrow F(\Pi^*)$ + best_move_value;
    }
} while (best_move_value < 0);
```

Fig. 1. A simple hill-climbing heuristic.

tion (the conditions that make it possible to specify that a move is tabu);
2. To identify the actual *tabu restrictions* based on the attributes;
3. To identify an effective *data structure* for updating the tabu status of moves; and
4. To identify *aspiration level criteria* allowing the tabu status of a move to be overridden under appropriate circumstances.

In addition to these elements, a *short-term memory function* is used to determine how long a tabu restriction will be enforced. Also, a *long-term memory function* is constructed in such a way as to allow the investigation of a number of alternative "fresh" starting points for the entire search procedure while encouraging selection of starting points not near those previously selected. The long-term memory function may be viewed as a means of creating an intelligent diversification of the regions to which the solution process is applied, rather than relying on a blind randomization process. (Additional roles of TS memory functions are described in [6].)

In the solution of P1, our main concern is to create a tabu status that prevents a move from being reversed while under the jurisdiction of the short-term memory, which we have chosen for P1 to be a specified number of future moves. This tabu status is modified by the application of aspiration criteria, as indicated below.

There are a variety of ways to create tabu restrictions within this context. For example, one can impose a tabu restriction such that, after a move of jobs $\pi(i)$ and $\pi(j)$, where $j > i$, job $\pi(i)$ cannot be placed *any* earlier, i.e., move to the "left," on subsequent schedules until the short-term memory has expired. The performance of the TS method under this restriction was not as good as it became under a more flexible tabu restriction. After initial experimentation with several options, we found the following tabu restriction to work well and embodied it in the rest of our investigation: after a move of jobs $\pi(i)$ and $\pi(j)$, where $j > i$, job $\pi(i)$ cannot be placed at or any earlier than position *i* on subsequent schedules until the short-term memory tenure for this job has expired. The attribute of a move to be recorded in order to implement this restriction becomes $\pi(i)$, the index of the job in position *i* prior to the move, and *i*, the position from which job $\pi(i)$ moved to the "right" to occupy position *j*. Accompanying this, we designed data structures to keep tract of moves that are classified as *tabu* and to free those moves from their tabu condition when their short-term memory tenure has expired. The data structures used in the solution of P1 are composed of a *tabu state, tabu list,* and *tabu position* that contain the following information:

tabu_ state ($\pi(i)$) = number of times job $\pi(i)$ appears in the tabu list, $\pi(i) = 1, \ldots, N$.
tabu_ list(ltabu) = $\pi(i)$, if job $\pi(i)$ is prevented from moving at or "left" of its tabu position (ltabu = $1, \ldots,$ tabu_ extent).
tabu_ position($\pi(i)$) = tabu position for job $\pi(i)$, $\pi(i) = 1, \ldots, N$.
aspiration_ level($\pi(i)$) = aspiration level for job $\pi(i)$, $\pi(i) = 1, \ldots, N$.

If tabu_ state($\pi(i)$) > 0, $\pi(i)$ has been moved to the right during the previous tabu_ extent moves and will reside at least once on the tabu_ list. Hence, the tabu _state vector removes the need to sequentially traverse the tabu list. In a move of jobs $\pi(i)$ and $\pi(j)$, if tabu_ state($\pi(j)$) > 0 and $i \leq$ tabu_ position($\pi(j)$), then the move is a tabu move. The tabu_ list keeps track of the jobs that are being prevented from moving at or "left" of their tabu position, as a way to record tabu restrictions. The argument of tabu_ list, ltabu (ltabu = $1, \ldots,$ tabu_ extent), is a pointer to the next available position in the circular tabu list of length, tabu_ extent. The tabu_ position contains the most recent tabu positions of jobs in the tabu list; i.e., if a job appears twice in the tabu list, only the most recent tabu position is considered. This structure implicitly enforces the short-term memory function by assigning a tabu status to a move for only the tabu_ extent most recent moves. We discuss the above structure in more detail in section 4.

Two *aspiration-level* tests are used in our application of the TS method to P1, as means of overriding tabu status. Both tests allow tabu moves to be executed but differ in the level of flexibility that they introduce to the search. The less flexible criterion, aspiration criterion A, allows a tabu move to be performed, i.e., allows a tabu job to move to a position earlier than its tabu position, during its short-term memory tenure, if

$$F(\Pi) + \text{move\_\_ value} < F(\Pi^*).$$

A less restrictive test, aspiration criterion B, allows a tabu move to be performed if

$$F(\Pi) + \text{move\_\_ value} < \text{aspiration\_\_ level } (\pi(j))$$

where the aspiration level for each job, aspiration__ level($\pi(i)$), is initially set to a large value and subsequently modified in the following way. Assume the move that repositions jobs $\pi(i)$ and $\pi(j)$ has the best move value for the current Π. Then the modification is performed by the following rule:

```
if (aspiration__ level($\pi(i)$) > $F(\Pi)$)
  aspiration__ level($\pi(i)$) = $F(\Pi)$
  if (aspiration__ level($\pi(j)$) > $F(\Pi)$)
  aspiration__ level($\pi(j)$) = $F(\Pi)$
```

4. A Prototype Tabu Search Method

In this section we identify how our prototype TS method operates to take advantage of the data structures introduced in section 3. We begin by describing the method in overview (see figure 2),

```
Initialize long term memory function;
$F(\Pi^*) \leftarrow \infty$;
do {
    Generate starting solution $\Pi$;
    Evaluate $F(\Pi)$;
    Initialize move_value matrix;
    Initialize short term memory function;
    do {
        Update long term memory function;
        best_move_value $\leftarrow \infty$;
        for (all candidate moves) {
            if (candidate move is admissible) {
                if (move_value < best_move_value) {
                    best_move_value $\leftarrow$ move_value;
                    best_move $\leftarrow$ current move;
                }
            }
        }
        Execute best_move;
        $F(\Pi) \leftarrow F(\Pi)$ + best_move_value;
        Update move_value matrix;
        Update short term memory function;
        if ($F(\Pi) < F(\Pi^*)$) {
            $\Pi^* \leftarrow \Pi$;
            $F(\Pi^*) \leftarrow F(\Pi)$;
        }
    } while (moves without improvement < max_moves);
} while ($F(\Pi^*)$ has changed in last 4 starting points);
```

Fig. 2. A prototype tabu search method for P1.

followed by a detailed specification of its operations. In broad terms, our TS method for P1 seeks an optimal schedule by making a succession of pairwise exchanges, or moves, to transform one schedule into another. The method starts by initializing the long-term memory function and generating an initial feasible heuristic solution, which is saved as the best solution found so far. The search is restarted from at least three more different starting solutions that are generated using the penalties assigned by the long-term memory function. Every time the search is restarted, the short-term memory function is reinitialized.

Given a starting solution, the method performs moves until a specified number of moves (max_ moves) elapses without obtaining a solution better than the best solution found on the current pass. After any change in the schedule, the long-term memory function is updated. The move to be made at a given iteration is found by calculating the move value of all *candidate* moves for the current solution. A move is considered to be a candidate if the jobs being exchanged are within a specified distance (number of positions). Since we are minimizing, the best candidate move possesses the algebraically smallest move value. More precisely, the best move is selected from the set of *admissible* candidates. A candidate is admissible either if it is not tabu or if its tabu status can be overridden by the aspiration criterion. The best move is then performed and the tabu data structure is updated. The best solution, Π^*, is updated if the current solution's $F(\Pi)$ is less than $F(\Pi^*)$. The procedure is repeated until the termination criterion is met, i.e., until the last four starting solutions fail to improve the best solution found so far.

4.1. Long-Term Memory Function

The long-term memory function we designed for P1 is embodied in an NxN matrix that contains the number of times that each job has been scheduled in each feasible position; i.e., the (i,j) element in the matrix contains the number of iterations that job i has been scheduled in position j. This function is used to generate a new starting solution after the search has been unable to find a better solution from the current starting point. Jobs are penalized according to the proportion of time spent in each position. The more time a job spends in a particular position, the larger the penalty assigned. This forces the heuristic that generates the starting point to find a "good" solution that avoids the scheduling of jobs in positions frequently occupied in the past. Initially all the elements in the matrix are zero. The long-term memory function is updated after every move in the following way:

Let the current schedule be $\Pi = \{0, \pi(1), \pi(2), \ldots, \pi(N), N+, 1\}$; then N elements in the matrix are changed as follows:

$$\text{long\_term}(\pi(i), i) = \text{long\_term}(\pi(i), i) + 1 \quad \text{for } i = 1, \ldots, N.$$

The proportion of time that each job $\pi(i)$ has spent in position j is found by dividing the content of each element $(\pi(i), j)$, for $j = 1, \ldots, N$, by the current numbers of moves.

4.2. Starting Solutions

Two methods of generating starting solutions, heuristics 1 and 2 from Vanston [12], are implemented in the application of the TS method to P1. While heuristic 2 gives consistently "better" solutions than heuristic 1, neither heuristic has proven to be uniformly superior for the TS method in terms of the number of moves required to reach the optimal solution. Both are one-pass heuristics based on the traveling salesman "nearest unvisited city" rule. The procedure starts by scheduling job 0 in position 0. Then the available job j that minimizes the distance from the previous job i is scheduled at the next position. The distance measure is a linear function of the setup cost, $s_{ij,}$ and the delay penalty "price" of scheduling job j before fellow available candidates. The difference between the two heuristics is in the distance measure. In heuristic 1 the distance measure is such that it fails to produce a natural-order schedule in the absence of setup charges.

When the heuristics are used to restart the search, a penalty that depends on the long-term memory function is added to the distance measure. In this way, the distance measure will increase if a job is being considered for a position it has occupied in the past.

4.3. Short-Term Memory Function

The short-term memory function is managed in our approach by the tabu state and the tabu list. The aspiration level and the tabu position of each job are also recorded and handled in parallel. The short-term memory function is initialized every time the search is started from a new starting solution. The initialization is as follows:

```
tabu_ state($\pi$(i)) = 0 for $\pi$(i) = 1, . . . , N.
tabu_ state(N + 1) = tabu_ extent.
tabu_ list(ltabu) = N + 1 for ltabu = 1 . . , tabu_ extent.
tabu_ position($\pi$(i)) = 0 for $\pi$(i) = 1, . . ., N.
ltabu = 0
```

The short-term memory function is then updated after every move. Let $\pi_B(i)$ and $\pi_B(j)$ be the jobs with the best move value found by the best_ move function. Then the following operations are required to update the tabu status of each job:

```
ltabu = ltabu + 1
if ltabu > tabu_ extent then ltabu = 1
tabu_ state(tabu _list(ltabu)) = tabu_ state(tabu_ list(ltabu)) - 1
tabu_ list(ltabu) = $\pi_B$(i)
tabu_ state($\pi_B$(i)) = tabu_ state($\pi_B$(i)) + 1
tabu_ position($\pi_B$(i)) = i
```

4.4. Best Move Function

Given a current solution, the *best move* is the admissible move that has the minimum or *best move value*. In order to find the best move, the best_ move function evaluates the move values of all the *candidate* moves. A move of jobs $\pi(i)$ and $\pi(j)$ becomes a candidate if $j - i \leq d$, where d is the maximum moving distance allowed. The maximum moving distance, d, has been set equal to $\lfloor N/2 \rfloor - 1$ for problems with up to 30 jobs, where $\lfloor x \rfloor$ is the largest integer less than or equal to x. For larger problems (N > 30), we allowed d to start smaller (to increase processing speed) and to be gradually increased to its specified value by the following formula:

$$d = \frac{(\lfloor N/2 \rfloor - 1)c}{4} \qquad \text{for } c = 1, \ldots, 4.$$

The value of c is originally set to 1 and is incremented by one, until 4 is reached, every time that the search is restarted after the best solution was not improved from the previous starting point. This dynamic change of d significantly reduces the time to the best solution of the larger problems. A candidate move becomes an *admissible* move if such a move is not tabu, or if it is tabu and the aspiration criterion is able to override its tabu status. The best_ move function records only the best admissible move and its corresponding move value.

A critical part involved in finding the best move value is the evaluation of the move values for all candidate moves. For problems with 15 and 20 jobs, there are 63 and 135 candidate moves, respectively; and in general, there are $3(d^2 - d)/2$ candidate moves in a given schedule. The move value can be calculated (without actually executing the move) far more efficiently than by direct application of its definition as follows:

Consider a move involving jobs $\pi(i)$ and $\pi(j)$, where $i < j$. Then the difference in setup cost, ΔS, between the value of the objective function after the move, $F(\bar{\Pi})$, and the value of the objective function before the move, $F(\Pi)$, is

$$\Delta S = s_{\pi(i-1)\pi(j)} + s_{\pi(j)\pi(i)} + s_{\pi(i)\pi(j+1)} - s_{\pi(i-1)\pi(i)} - s_{\pi(i)\pi(j)} - s_{\pi(j)\pi(j+1)} \quad \text{if } j = i + 1.$$

$$\Delta S = s_{\pi(i-1)\pi(j)} + s_{\pi(j)\pi(i+1)} + s_{\pi(j-1)\pi(i)} + s_{\pi(i)\pi(j+1)} - s_{\pi(i-1)\pi(i)} - s_{\pi(i)\pi(i+1)} - s_{\pi(j-1)\pi(j)} - s_{\pi(j)\pi(j+1)} \quad \text{if } j \neq i + 1.$$

The move value is given by

$$\text{move\_value} = \sum_{k=i}^{j-1} t_{\pi(k)}$$
$$(p_{\pi(i)} - p_{\pi(j)}) + \sum_{k=i}^{j-1} p_{\pi(k)} (t_{\pi(j)} - t_{\pi(i)}) + \Delta S$$

With this streamlined calculation, the evaluation of all candidate moves at each iteration is still somewhat expensive, and in fact it is possible to do better by additionally introducing a memory scheme that isolates and updates only the relevant evaluations that change from iteration to iteration. We accomplish this by creating a move_value matrix that contains the move value of all candidate moves, i.e., the (i,j) element in the matrix contains the move value of the swap move that exchanges the jobs in position i and j of the current schedule. This matrix is initialized every time a new starting point is generated. The initialization process consists in evaluating the move value of all candidate moves for the current schedule. The matrix is updated at every iteration as follows:

Let $\pi(r)$ and $\pi(q)$ be the jobs repositioned on the previous iteration, and let the exchange of jobs currently in position i and j, i.e., jobs $\pi(i)$ and $\pi(j)$, be a candidate move. Then, we calculate the new move value if and only if i or j is between $[r - 1, q + 1]$, and otherwise use the old move value.

The updating procedure is designed to identify the subset of candidate moves whose move values have been affected by the execution of the best move in the previous iteration. Members of this subset are the only ones subjected to updating, while the rest of the candidate moves keep their old move values. The introduction of the move value matrix causes a significant reduction in CPU time, e.g., requiring on average 62% less time to approximately solve 35-job problems than without its use.

5. An Illustrative Example

In this section we illustrate the TS method using the five-job problem used by Barnes and Vanston [2] to exemplify their BB and hybrid DPBB approaches to the problem. The data for the problem is given in tables 1 and 2.

Table 1. Job parameters

Job	Duration (days)	Delay penalty ($/day)	Ratio
1	3	700	3/700
2	4	800	1/200
3	1	100	1/100
4	4	300	4/300
5	5	200	5/200

Table 3 summarizes the tabu search process when the following options are used to start and direct the search:

Maximum number of moves without improvement: 2
Maximum moving distance: 1
Starting solution: Heuristic 1
Aspiration criterion: A
Short term memory length: 3

The entries of this table make it possible to trace in detail each iteration of the approach.
Since the last four starting points resulted in the same best objective function value, the procedure stopped at the twelfth iteration. The best solution found is in fact the optimal solution.

6. Preliminary Computational Experience

A C language computer code was written for the prototype TS method designed for P1. Three sets of problems of different problem sizes have been used to test the performance of the TS method. A set of 20-job problems was generated and used by Barnes and Vanston [2] to test their BB1, BB2, and DPBB algorithms. The processing times, delay penalties, and setup charges were randomly generated from uniform distribu-

Table 2. Setup costs

		j					
s_{ij}		1	2	3	4	5	6
	0	1100	600	1200	2000	1400	∞
	1	∞	1300	700	1200	1100	1000
i	2	900	∞	1100	1300	600	1200
	3	900	1000	∞	2000	700	1500
	4	1000	700	800	∞	600	1200
	5	1400	1300	1200	1300	∞	900

Table 3. Tabu search for a five-job problem

Iteration	Current schedule	Current objective	Best move	Move value	Tabu state	Tabu list	Tabu position	Best objective
0*	(3,1,2,4,5)	14900			(0,0,0,0,0,3)	(6,6,6)	(0,0,0,0,0)	
1	(3,1,2,4,5)	14900	(3,1)	−1000	(0,0,1,0,0,2)	(3,6,6)	(0,0,1,0,0)	14900
2	(1,3,2,4,5)	13900	(3,2)	1000	(0,0,2,0,0,1)	(3,3,6)	(0,0,2,0,0)	13900
3	(1,2,3,4,5)	14900	(1,2)	−900	(1,0,2,0,0,0)	(3,3,1)	(1,0,2,0,0)	
4*	(2,4,1,3,5)	15500	(4,1)	−2000	(0,0,0,1,0,2)	(4,6,6)	(0,0,0,2,0)	
5	(2,1,4,3,5)	13500	(4,3)	500	(0,0,0,2,0,1)	(4,4,6)	(0,0,0,3,0)	13500
6	(2,1,3,4,5)	14000	(1,3)	0	(1,0,0,2,0,0)	(4,4,1)	(2,0,0,3,0)	
7*	(3,2,1,5,4)	16500	(5,4)	−1600	(0,0,0,0,1,2)	(5,6,6)	(0,0,0,0,4)	
8	(3,2,1,4,5)	14900	(3,2)	−900	(0,0,1,0,1,1)	(5,3,6)	(0,0,1,0,4)	
9*	(3,1,2,5,4)	15900	(3,1)	−1000	(0,0,1,0,0,2)	(3,6,6)	(0,0,1,0,0)	
10	(1,3,2,5,4)	14900	(5,4)	−1000	(0,0,1,0,1,1)	(3,5,6)	(0,0,1,0,4)	
11*	(3,1,4,2,5)	16200	(4,2)	−1300	(0,0,0,1,0,2)	(4,6,6)	(0,0,0,3,0)	
12	(3,1,2,4,5)	14900	(3,1)	−1000	(0,0,1,1,0,1)	(4,3,6)	(0,0,1,3,0)	

*Current solution is a new starting point.

tions with $1 \leq t_i \leq 12$, $30 \leq p_i \leq 110$, and $500 \leq s_{ij} \leq 1500$. Three sets of five test problems with 25, 30, and 35 jobs were constructed using these same distributions and ranges. Each of the 20-job problems has been run using different parameters: two starting points (heuristic 1 and 2), two aspiration criteria (A and B), and three short-term memory function sizes (5, 6, 7 and 6, 7, 8 for aspiration criterion A and B, respectively). A maximum of 200 moves without improvement has been used as a cutoff criterion to terminate each run. The TS method found the optimal solution of all the problems, before the termination criterion was met, for at least one of the settings.

The experiments with larger problems ($N > 20$) were conducted restricting the size of the short-term memory function to 6 and 7, in two different sets of tests. For each problem a lower bound was obtained by calculating, separately, the delay penalty and the setup cost portions of the objective function. The delay penalty portion was found by ignoring the setup costs and ordering the jobs in their natural order. The setup cost portion was calculated by ignoring the processing times and delay penalties and solving the assignment problem.

The first four columns of table 4 show the problem number and the average, worst, and best solutions found by the TS method for the test problems. The average solution is calculated by adding the values of the objective function obtained by each setting and dividing the sum by the total number of settings. For the 20-job problems 8 and 9, every possible setting obtained the optimal solution. The fifth column shows a lower bound obtained by solving an assignment problem [12]. The sixth column shows the value of the proportional deviation of the best solution from the lower bound for each problem. The proportional deviation is calculated by dividing the absolute deviation (differences between the best solution and the lower bound) by the value of the lower bound. The last three columns show the average time to the best solution, the average time to the completion (end) of the search, and the total time required to run all settings. The times are given in CPU seconds of an IBM PS/2 Model 80 microcomputer. From the information shown in table 4, the following observations can be made about the performance of the TS method:

- The average proportional deviation of the best objective value found from its calculated lower bound, when the best value is in fact known to be optimal (for the 20-job problems), is similar to this same average proportional deviation for problems where an optimum is unknown (with $N > 20$). Assuming the percentage gap between the lower bound value and the optimum value does not diminish with problem size (the opposite would seem more likely), the proximity of the values obtained to the optimal values is very close.
- The average and worst solutions indicate that most of the solutions obtained by the TS method are closer, in value, to the best solution

Table 4. Summary of computational experience for PI

Problem	Average	Solutions worst	Best	Lower bound	Proportional deviation	Best	Times end	Total
20-Job								
1	86,995	87,086	86,980*	82,932	0.049	7.5	15.7	188
2	79,194	79,363	79,134*	75,120	0.053	3.4	11.3	136
3	98,086	98,163	98,059*	95,680	0.025	5.8	14.2	170
4	83,741	83,823	83,731*	80,569	0.039	4.5	12.8	153
5	98,721	98,755	98,675*	94,374	0.045	3.2	11.7	140
6	90,975	91,301	90,919*	87,631	0.037	8.8	17.1	205
7	81,256	81,410	81,241*	77,307	0.051	2.4	10.3	123
8	81,198*	81,198*	81,198*	77,979	0.041	3.1	11.8	141
9	87,283*	87,283*	87,283*	84,000	0.039	3.3	11.2	134
10	94,237	94,365	94,192*	90,300	0.043	4.7	13.4	161
				Average:	0.042	4.7	12.9	155
25-Job								
1	87,042	87,226	86,799	82,610	0.051	6.6	19.8	158
2	111,233	111,353	111,182	107,108	0.038	11.6	23.3	186
3	90,760	91,031	90,657	86,628	0.046	8.0	20.1	161
4	107,981	108,154	107,859	102,573	0.051	10.9	22.9	183
5	77,384	77,571	77,274	72,664	0.062	8.1	20.0	160
				Average:	0.050	19.1	21.2	170
30-Job								
1	105,163	105,493	104,785	99,777	0.050	6.8	24.0	192
2	138,613	138,864	138,414	131,829	0.050	9.0	25.6	205
3	129,197	129,481	128,883	124,656	0.034	8.3	26.0	208
4	105,827	106,332	105,514	99,834	0.057	8.9	26.5	212
5	115,780	115,919	115,625	109,337	0.057	11.0	27.4	219
				Average:	0.050	8.8	25.9	207
35-Job								
1	177,995	178,459	177,739	171,554	0.036	11.0	26.8	214
2	227,889	228,079	227,530	220,901	0.030	7.9	22.4	179
3	179,713	180,208	179,404	173,897	0.032	10.8	27.0	216
4	165,621	166,128	165,192	158,819	0.040	14.1	29.8	238
5	164,370	164,577	163,965	157,679	0.039	5.3	17.6	141
				Average:	0.035	9.8	24.7	198

*Optimal solution.

than to the worst one, and even the worst solution is frequently within 0.4% of the best.

- The dynamic change of the maximum moving distance allows the problems with 35 jobs to be solved without increasing the average total computational time. (This total CPU time becomes 339 seconds when a constant moving distance is used).

The TS method succeeds in solving reasonably large problems to near optimality (as inferred from the lower-bound comparison). The computation time for each solution trial is relatively small (an average of 24.7 seconds for 35-job problems), but the fact that a number of parameter settings are used results in a somewhat large total computational effort. We must also add that the average proportional deviations from the lower bounds of the initial solutions generated with heuristics 1 and 2 are 0.091 and 0.063, respectively.

7. An Improved TS Method

In order to further reduce the total computational effort linked with obtaining a solution to each problem, an additional strategy to move from one solution to another was incorporated into our prototype TS method. This strategy consists of introducing a move that transfers one job from its current position to an earlier or later position in the schedule. The direct consequence of adding this class of moves (referred to as *insert* moves) to the local search for the best move is an increase of the computational burden associated with each iteration. However, it was expected that the added searching power would allow the TS method to obtain solutions of the same or better quality than before, starting from any given initial solution.

This speculation proved correct. In fact, the modified TS method that incorporated insert as well as exchange moves proved so successful that it became unnecessary to restart the method from alternative initial solutions. We selected the same set of parameter values found most consistent in our previous experiments, i.e., the one that most often yielded the best solution to a problem. This set of parameters employs heuristic 2 to provide the starting solution, aspiration level criterion A, and a tabu size of 7. Moreover, with the speedup provided by using only a single starting point, we did not bother to start *d* at a smaller value and increment it for problems with $N > 30$, but set *d* to $\lfloor N/2 \rfloor - 1$ for problems of all sizes. The new termination criterion used stopped the search after a predetermined total number of iterations (500 for our experiment). The results of this experiment can be summarized as follows:

- The average CPU time per solution attempt was 16.6, 22.8, 31.4, and 40.9 seconds for problems with 20, 25, 30, and 35 jobs, respectively.
- The average CPU time to obtain the best solution found in each attempt was 4.8, 7.4, 9.1, and 19.0 seconds, respectively, for these problem sizes.
- Optimal solutions were found for all but one of the 20-job problems. (The best solution found for problem 2 deviated from optimality by 0.04%.)
- Better solutions were found by the modified method for the following problems: 25-job problem 5 (77,185), 30-job problems 2 (138, 301) and 5 (115,600), and 35-job problems 1 (177,333), 2 (227,121), 3 (179,340), and 5 (163,602).
- For the larger problems ($N > 20$), the modified method did not give solutions as good as the original method in only three instances: the 30-job problems 2 (138,440) and 4 (105,568), and 35-job problem 4 (165,243).

Furthermore, since the improved method does not include a restarting mechanism, we can measure the improvement achieved over the initial solution (i.e., the one generated by heuristic 2). On the average, the best solution found by the improved TS is 98% of the initial solution. Our success in combining the insert and swap moves within the TS framework raises the question of whether this combination might also work well independently, i.e., by simply employing a hill-climbing procedure with these moves that restarts multiple times. To test this, we also compared our method against a refinement of the multiple-start hill-climbing procedure known as GRASP [13], which seeks improved starting candidates by a randomized adaptive greedy process. Our approach obtains significantly better solutions overall in considerably less time. A fuller report of this experimentation and its outcomes appears in [14].

In order to test the ability of our method to obtain high quality solutions in an alternative setting, the TS procedure was adapted to handle a more complex version of problem P1. This new problem can be represented mathematically as follows:

$$\text{Minimize } F(\Pi) = C(\Pi) + S(\Pi)$$

where

$$C(\Pi) = \sum_{i=1}^{N} c_{\pi(i)}\, p_{\pi(i)} \qquad \text{(P2)}$$

and

$$C_{\pi(i)} = \sum_{j=1}^{i} t_{\pi(j)} + a_{\pi(j-1),\, \pi(j)}$$

In this model, a_{ij} is the time required for the machine to be changed from processing job i to processing job j, i.e., the setup time between job i and j. Also, the delay penalties are charged for every unit of time that the completion of the jobs is delayed from time zero. The non-identical, multiple-machine version of this problem was addressed by Vanston in [12]. Since his DPBB procedure was limited to very small instances of the more general case, he conducted experiments using problem instances with up to 24 jobs for the single-machine case. (The method is incapable of handling problems with 25 jobs or larger.) We used this set of nine problems to perform an additional and final computational experiment.

Although the substantial difference in the sizes of problems capable of being addressed by tabu search and the DPBB optimizing method already argues for the practicality of the TS approach, we sought to determine the relative efficiencies of these approaches for the problems the DPBB method was able to handle. Our experiment consists of a single solution attempt of 500 iterations for each problem. A modified heuristic 2 was used to provide the initial solutions, and the short-term memory was set to ⌊ N/2 ⌋ moves for problems with up to 12 jobs and to 7 moves for larger problems. Table 5 presents a comparison of the solutions obtained and CPU times required by the optimizing algorithms BB and DPBB and the approximate procedure TS. (This comparison does not include the performance of a DP algorithm, since this approach was abandoned in [12] after proving inferior to BB and DPBB for small problems.) The CPU times (in seconds) reported for BB and DPBB were achieved with FORTRAN implementations of these algorithms in a CDC 6600 using the MNF compiler. (The TS method, as before, was implemented in C on an IBM PS/2 Model 80 microcomputer.)

As shown in table 5, the computational effort involved in solving the test problems by means of BB and DPBB increases rapidly with the problem size. The BB algorithm reaches the time limit of 1000 seconds, imposed by Vanston in [12], without confirming optimality for problems with 20, 22, and 24 jobs. (The most recent upper bound found by this approach unfortunately was not reported, as indicated in table 5. Hence we do not know if this bound was as good as that obtained by the tabu search solutions.) DPBB is able to solve, within the time limit, problems with up to 22 jobs, but it terminates with a memory overflow while attempting the solution of the 24-job problem. At this point the most recent upper bound of 252,887 is inferior to the TS solution of 246,821. It can be also observed that the time to reach the termination criterion of 500 iteration for the TS method increases with the number of jobs, but the increments are not as significant as those experienced by the optimizing techniques. The times to obtain the best solution found by the TS method show that, although TS is instructed to execute 500 moves on all problems,

Table 5. Summary of computational experience for P2

	Solution			CPU seconds				
					DPBB†		TS††	
Problem size	BB	DPBB	TS	BB†	Total time	Time to best	Chosen limit	Time to best
8	47,544*	47,544*	47,544*	0.2	0.8	0.18	9.56	0.00
10	67,563*	67,563*	67,563*	0.9	2.0	0.36	14.78	0.11
12	76,656*	76,656*	76,656*	2.8	6.2	1.74	21.70	0.44
14	96,135*	96,135*	96,135*	10.4	8.6	2.41	29.11	0.49
16	141,130*	141,130*	141,130*	67.4	47.4	13.27	34.16	0.33
18	191,320*	191,320*	191,320*	194.2	187.0	52.36	44.38	32.46
20	n. r.	185,459*	185,459*	>1000	916.2	256.54	55.58	3.46
22	n. r.	213,756*	213,756*	>1000	417.9	117.01	64.60	11.10
22	n. r.	252,887ᵘ	246,821	>1000	m. o.	—	72.06	6.92

†CDC 6600.
††IBM PS/2 Model 80.
*Optimal solution.
ᵘMost recent upper bound.
n. r. Not reported.
m. o. Memory overflow.
(If adjusted for relative speeds of the computers used, the times for the TS method would be approximately twice as fast as those shown.)

the best solutions were generally found early in the search (i.e., somewhat before the first 100 iterations). With the exception of the 18-job problem, all other solutions are found during the first 12 seconds of search. By contrast, the DPBB procedure is reported to require an average 28% of its total solution time to reach the best solution obtained, which results in a significant difference in the "Time to Best" columns for these methods shown in table 5. To have a better base for comparing the CPU times across machines, we have empirically determined that the CDC 6600 mainframe computer used by the DPBB approach operates at least twice as fast as the IBM PS/2 Model 80 microcomputer used by the TS approach. This was achieved by comparing the execution time of a code originally implemented in the CDC 6600, which has been recently adapted for the IBM PS/2 Model 80.

Table 5 also shows the consistently high quality of the solutions obtained by the TS method. This approach not only found the optimal solutions to all problems for which these solutions are known, but also it was able to improve the best-known solution to the 24-job problem by 6,066 units (i.e., a 2.4% reduction on the previous best-known value).

8. Final Remarks

Our experimentation shows that the TS method we have designed for the single-machine scheduling problems P1 and P2 is remarkably effective. Solution quality rivals and generally surpasses that of optimizing methods tailored for this problem, under conditions where the optimizing methods are allowed to consume much greater amounts of time (not only to verify optimality, but to obtain the best solution found). In addition, our application of tabu search employs much less memory and is capable of generating high-quality solutions very quickly for problems with $N > 20$, which the optimizing methods are incapable of handling within either reasonable time or reasonable memory restrictions.

We have shown that the the TS method originally designed for P1 was successfully adapted to provide solutions to the related problem P2. In the same way, this procedure can be modified to approximately solve other single machine scheduling (SMS) problems with different objective functions. As presented by Gupta and Kyparisis [15], the SMS problem can be classified on the basis of objective functions and problem constraints. Simple objective functions based on completion times include total completion time, mean completion time, total weighted completion time, and maximum completion time. Typical objective functions considered for problems with due dates include total tardiness, mean tardiness, mean lateness, total weighted tardiness, maximum tardiness, number of tardy jobs, and weighted number of tardy jobs. Composite objective functions (like the one used in P1 and P2) can be formed by combining single objective functions. By changing only the generation of starting solutions and the definition of the move value, the general structure of the TS in its current form can be used when other types of objective functions and constraints are incorporated within the SMS problem. Considerations for future research with applications to additional settings are found in [6].

Acknowledgment

This research was supported in part by the Joint Air Force of Scientific Research and Office of Naval Research Contract No. F49620-90-C-0033 at the University of Colorado.

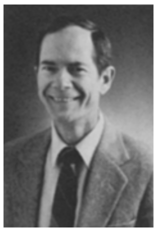
References

1. S.E. Elmaghraby and S.H. Park, "Scheduling jobs on a number of identical machines, *AIIE Trans.*, vol. 16, no. 1, pp. 1–13, March 1974.
2. J.W. Barnes and L.K. Vanston, "Scheduling jobs with linear delay penalties and sequence dependent setup costs," *Operations Res.*, vol. 29, no. 1, pp. 146–160, January–February 1981.
3. T.L. Morin and R.F. Marsten, "Branch-and-bound strategies for dynamic programming," *Operations Res.*, vol. 24, no. 4, pp. 611–627, July–August 1976.
4. D. de Werra and A. Hertz, "Tabu search techniques: A tutorial and an application to neural networks, *OR Spectrum*, vol. 11, pp. 131–141, 1989.
5. F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, July–August 1990.
6. F. Glover and M. Laguna, "Tabu search," in *Modern Heuristics for Combinatorial Optimization*, C.R.

Reeves (Ed.), Blackwell Scientific Publications, Oxford, 1993.
7. A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *Computing,* vol. 29, pp. 345–351, 1987.
8. F. Glover, E. Taillard, and D. de Werra, "A user's guide to tabu search," *Ann. Operations Res.* (in press).
9. I.H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Ann. Operations Res.* (in press).
10. J. Chakrapani and J. Skorin-Kapov, "Massively parallel tabu search for the quadratic assignment problem," *Ann. Operations Res.* (in press).
11. M. Widmer and A. Hertz, "A new method for the flow sequencing problem," *Eur. J. Operations Res.,* vol. 41, pp. 186–193, 1989.
12. L.K. Vanston, "A hybrid dynamic programming/branch-and-bound algorithm to solve multiple-machine scheduling problems," Ph.D. Dissertation, Mechanical Engineering Department, The University of Texas at Austin, 1979.
13. T.A. Feo and M.G.C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Res. Lett.,* vol. 8, pp. 67–71, 1989.
14. M. Laguna, J.W. Barnes, and F. Glover, "Tabu search methods for a single machine scheduling problem," *J. Intell. Manufact.,* vol. 2, pp. 63–74, 1991.
15. S.K. Gupta and J. Kyparisis, "Single machine scheduling research," *OMEGA Int. J. Mgmt. Sci.,* vol. 15, no. 3, pp. 207–227, 1987.

Manuel Laguna is an Assistant Professor of Management Science in the Graduate School of Business Administration of the University of Colorado at Boulder. He received master's and doctoral degrees in Operations Research and Industrial Engineering from the University of Texas at Austin. Dr. Laguna has done extensive research in the interface between artificial intelligence and operations research to develop solution methods for problems in the areas of production scheduling, telecommunications, and facility layout. As part of the University of Colorado and US West Partnership, Dr. Laguna collaborates with the Advanced Knowledge Systems Research Group of US West Advanced Technologies. He is Editor of the special issue on tabu search of *Annals of Operations Research* and a member of the Operations Research Society of America and the International Honor Society Omega Rho.

J. Wesley Barnes, Cullen Trust for Higher Education Endowed Professor in Engineering, is the Coordinator of the Graduate Program in Operations Research and Industrial Engineering at the University of Texas at Austin. He is a past Associate Editor of the *Transactions of the Institute of Industrial Engineers* and is a Registered Professional Engineer in the State of Texas. He is the author of several books, among which are *Network Flow Programming,* winner of the Institute of Industrial Engineers Book-of-the-Year Award for 1980, and *Statistical Analysis for Engineers, a Computer-Based Approach,* Prentice Hall Publishing, 1988.



Fred Glover is the US West Chaired Professor in Systems Science at the University of Colorado, Boulder. He has authored or co-authored more than 200 published articles in the fields of mathematical optimization, computer science, and artificial intelligence, with particular emphasis on practical applications in industry and government. In addition to holding editorial posts for journals in the U.S. and abroad, Dr. Glover has been featured as a National Visiting Lecturer by the Institute of Management Science and the Operations Research Society of America and has served as a host and lecturer in the U.S. National Academy of Sciences Program of Scientific Exchange.

Professor Glover is the recipient of numerous awards and honorary fellowships, including those from the American Association for the Advancement of Science, the NATO Division of Scientific Affairs, the Institute of Management Science, the Operation Research Society, the Decision Science Institute, the U.S. Defense Communications Agency, the Energy Research Institute, the American Assembly of Collegiate Schools of Business, Alpha Iota Delta, and the Miller Institute for Basic Research in Science. He serves on the advisory boards of several organizations and is co-founder of Optimization Technologies, Inc., Analysis and Research and Computation, Inc., and the nonprofit research organization Decision Analysis and Research Institute.