Journal of the Operational Research Society (1996) 47, 1054–1064 0160-5682/96 $12.00

Single Machine Scheduling with Major and Minor Setup Times: A Tabu Search Approach

E. NOWICKI and S. ZDRZAŁKA
Technical University of Wrocław, Poland

The paper deals with a single machine scheduling problem in which jobs are grouped into families that require major setup times, and where jobs within families require for processing minor setup times. The former are sequence independent and the latter have special triangular structure. The problem is to find a partition of families into batches, sequences of jobs in particular batches and a sequence of batches which minimize a general regular cost function. The tabu search algorithm for finding near-optimal schedules is proposed and results of computational experiments are presented.

Key words: sequencing, scheduling, batching, setup times, tabu search

INTRODUCTION

Scheduling and sequencing with batching and machine setups is the subject of growing interest in the recent literature dedicated to production planning, scheduling and sequencing, see Potts and Van Wassenhove[1]. This is due to the practical importance of this class of models and the fact that very few results have appeared in this area so far. In this paper, we consider one such scheduling model, motivated by a real production planning situation. Jobs, which have to be processed on a single machine, are grouped into subsets (families) such that a large (major) setup time is required when there is a switch from a job of one family to a job of another family. Small (minor) setup times are incurred between processing the jobs of the same family. Major setup times are sequence independent while the minor setup time has some positive value, the same for all jobs and all families, when there is a switch from the job with higher index to the one with lower one, and it is zero otherwise (a triangular structure). Processing the jobs of each family contiguously and in the order of increasing indexes minimizes the total setup time, but this need not be an optimal schedule with respect to other cost functions, e.g. the ones involving due dates and penalties for job latenesses. Generally, the problem requires decision to be taken on dividing the families into batches (batching), job sequences in particular batches and a sequence of batches so as to minimize a general regular cost function. Since all these decision variables are implicit in a permutation of jobs, we actually deal with a sequencing problem.

Although some special cases of the considered class may easily be solved exactly or approximately, see, e.g. Potts and Van Wassenhove[1], Monma and Potts[2], Zdrzałka[3,4], generally, the scheduling problems with batching and setups are essentially harder than their standard counterparts, and designing an exact algorithm that solves a problem of practical size in a reasonable time is rather unlikely. Therefore in this paper, we focus our attention on designing and examining the tabu search algorithm, the local search technique developed by Glover[5,6]. Recently reported results of applying tabu search to hard standard (without setups) scheduling problems show that it is the most effective method amongst the local search techniques, see Vaessens, Aarts and Lenstra[7]. For instance, fast and effective algorithms for parallel machine scheduling, flow shop and job shop problems have been presented in Glover and Hibscher[8] and Nowicki and Smutnicki[9,10] respectively.

We propose a general tabu search algorithm for solving our scheduling problem with major and minor setup times and general cost functions, and then show results of computational experiments for two cost functions: maximum weighted lateness and total weighted tardiness. The procedure

Correspondence: E. Nowicki, Institute of Engineering Cybernetics, Technical University of Wroclaw, ul Janiszewskiego 11/17, 50-372 Wroclaw, Poland

was tested on the problem instances taken from real production processes with the number of jobs ranging between 40 and 200.

PROBLEM FORMULATION AND PRELIMINARIES

There are n jobs, identified by the integers from the set $J = \{1, 2, \ldots, n\}$ which are to be processed on a single machine. Job j requires a processing time $p_j > 0$. Preemption is not permitted, and at any time the machine can execute at most one job. The set of jobs is partitioned into B subsets $I_1, \ldots, I_B$, called families. Processing job j after job i requires a setup time s_{ij} which has to be respected in a feasible schedule as follows. Whenever job i is processed immediately before job j, the processing of job j must be preceded by an idle time of length at least s_{ij}; this idle time can be regarded as a processing time of a setup job. When job j is processed first, then it has to be preceded by an initial setup with time s_{0j}; 0 can be regarded as an index of an initial job representing the initial status of the machine. The setup times take the following values:

- If $j \in I_b$ and i belongs to another family than I_b, or $i = 0$, then $s_{ij} = S_b$; S_b is a major setup time of family I_b, $1 \leqslant b \leqslant B$.
- If i and j belong to the same family, then $s_{ij} = 0$ if $i < j$, and $s_{ij} = s$ if $i > j$; s is called a minor setup time. The minor setup time s is the same in all families and $s < S_b$ for all b.

Let a permutation π of J determine the job processing order; $\pi(i)$ is the i-th job in π. We denote by Π the set of all permutations of J. Let C_j be the earliest, with respect to some processing order, completion time of job j. Clearly, for processing order π, the earliest time by which job $\pi(i)$ is completed is given by

$$C_{\pi(i)} = \sum_{k=1}^{i} (s_{\pi(k-1),\,\pi(k)} + p_{\pi(k)}),$$

where $\pi(0) \equiv 0$.

We consider the following general optimization problem. Let $f_j(C_j)$ be the cost of completing the job j in time C_j, $1 \leqslant j \leqslant n$. We assume that f_j are nondecreasing functions (so called regular cost measures). The cost of processing the jobs in the order given by π is $f(\pi)$, where $f \in \{f_{\max}, f_{\text{sum}}\}$ and $f_{\max}(\pi) = \max_{1 \leqslant i \leqslant n} f_{\pi(i)}(C_{\pi(i)})$, $f_{\text{sum}}(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)})$. The problem is to find a permutation $\pi^* \in \Pi$ which minimizes the objective function $f(\pi)$ on the set Π.

For instance, let $d_j \geqslant 0$ and $w_j \geqslant 0$ be the due date and the weight of job j, respectively. If $f_j(C_j) = w_j(C_j - d_j)$ and $f = f_{\max}$, then we have the problem of minimizing the maximum weighted lateness, and if $f_j(C_j) = w_j \max\{0, C_j - d_j\}$ and $f = f_{\text{sum}}$, the problem is to minimize the total weighted tardiness.

The problem with objective function $f_{\max}$ is the generalization of the case that has been shown by Bruno and Downey[11] to be NP-hard, thus it is also NP-hard. Its special case with $s_{ij} = 0$ for all i and j is solvable in polynomial time by using the algorithm of Baker *et al.*[12]. The problem with the objective function given in the form of f_{sum} is strongly NP-hard since its special case with $f_j(C_j) = w_j \max\{0, C_j - d_j\}$ and $s_{ij} = 0$ for all i and j, that is the standard single machine scheduling problem of minimizing the total weighted tardiness, is NP-hard in the strong sense, see Lenstra, Rinnooy Kan and Brucker[13]. Since both problems are NP-hard, we focus our attention on heuristic approaches: especially on the tabu search technique.

We now introduce further notation and show some useful properties of the setup time function s_{ij}.

For a permutation π, let u be the smallest position such that $f_{\max}(\pi) = f_{\pi(u)}(C_{\pi(u)})$. We call $(\pi(1), \ldots, \pi(u))$ a critical sequence and $\pi(u)$, a critical job of π. Further, let $g(j)$ be the index of the family to which job j belongs, $g(j) = b$ iff $j \in I_b$; $g(0) = 0$.

Lemma 1

For any triple of different jobs, i, j, k such that $i \in J \cup \{0\}$ and $j, k \in J$,

$$d_{ijk} \equiv s_{ij} + s_{jk} - s_{ik} \in D,$$

where

$$D = \{S_a + S_b: a < b,\ a, b \in \{1, 2. \ldots, B\}\}$$
$$\cup \{S_a + S_b - s: a < b,\ a, b \in \{1, 2, \ldots, B\}\}$$
$$\cup \{S_a: a \in \{1, 2, \ldots, B\}\} \cup \{0, s\}.$$

Proof of Lemma 1 is given in the Appendix.

The cardinality of D is $B(B-1)/2 + B(B-1)/2 + B + 2 = B^2 + 2$, however when $S_b = S$ for all b, then $|D| = 5$. It follows from Lemma 1 and the fact that $S_b > s$ for all b that $d_{ijk} \geqslant 0$ which, in view of the definition of d_{ijk}, implies that setup times s_{ij} satisfy the triangle inequality.

THE TABU SEARCH ALGORITHM

We present a tabu search algorithm, called *TS*, which is capable of handling any sequencing problem with the described structure of major and minor setup times and a general regular cost function. Roughly speaking, the tabu search technique[5,6] consists in starting from an initial basic permutation of n jobs and searching through its neighbourhood, a set of permutations related to the basic one, for a permutation with the lowest cost. Then the search repeats from this best one, as a new basic permutation, and the process is continued. In the search process, each permutation belonging to a neighbourhood is considered as a result of a move, an operation of changing the positions of a number of elements in the basic permutation. In order to avoid cycling, becoming trapped at a local optimum, or continuing the search in a too narrow region, a mechanism of a tabu list is introduced. The elements of the tabu list define moves that cannot be applied currently; that is they determine forbidden permutations in the currently analysed neighbourhood. The list content is refreshed each time the new basic permutation is found; the oldest element is removed and the new one is added. The search stops when a given number of iterations has been reached without improving the currently best solution has been reached, or the procedure has performed a given number of iterations, or the neighbourhood is empty, etc.

In practice, the design of particular components of a tabu search algorithm is considered as an art. Some of the components used in our algorithm, namely the form of tabu list, the tabu status of a move and the search strategy have been proposed by Nowicki and Smutnicki[9,10], where they were successfully applied to the job and flow shop problems. In this paper we extend these techniques, in the original or modified form, to the problem considered.

Moves and neighbourhood

Let $v = (x, y)$ be a pair of positions in a permutation π, $x, y \in \{1, \ldots, n\}$, $x \neq y$. With respect to a permutation π, the pair $v = (x, y)$ defines a move that consists in removing job $\pi(x)$ from position
x and inserting it in position y. Thus move v generates a permutation π_v from π in the following way:

$$\pi_v = (\pi(1), \ldots, \pi(x-1), \pi(x+1), \ldots, \pi(y), \pi(x), \pi(y+1), \ldots, \pi(n)) \quad \text{if } x < y,$$
$$\pi_v = (\pi(1), \ldots, \pi(y-1), \pi(x), \pi(y), \ldots, \pi(x-1), \pi(x+1), \ldots, \pi(n)) \quad \text{if } x > y.$$

The neighbourhood of π consists of permutations π_v generated by moves from a given set M. We define M in the following way. Let $M_x^L = \{(x, y): 1 \leqslant y \leqslant x-2\}$ be the set of moves to the left, let $M_x^R = \{(x, y): x+1 \leqslant y \leqslant n\}$ be the set of moves to the right, and let $M_x = M_x^L \cup M_x^R$ be the set of all moves from position x. When $f = f_{\max}$, we assume $M = (\bigcup_{j=1}^{u-1} M_j) \cup M_u^L \equiv M^{\max}$, where u is the position of the critical job in π, and for $f = f_{\text{sum}}$ we put $M = \bigcup_{x=1}^{n} M_x \equiv M^{\text{sum}}$. Note that $M^{\text{sum}} = \{(x, y): y \notin \{x-1, x\}\}$, $x, y \in \{1, 2, \ldots, n\}\}$.

Clearly, $|M_x^L| = \max\{x-2, 0\}$, $|M_x^R| = n - x$ and $|M_x| = n - \min\{x, 2\}$. Thus the set M^{sum} has cardinality $(n-1)^2$ while the cardinality of $M^{\max}$ is $(n-1)(u-1)$. Lower cardinality of $M^{\max}$ is obtained by deleting from the entire set some of the moves that do not immediately improve the value of $f_{\max}$. In the definition of $M^{\max}$ we take advantage of the following property.

Lemma 2

In the problem with objective function $f_{\max}$, for any permutation π with critical job $\pi(u)$ and any move $v \in (\bigcup_{j=u+1}^{n} M_j) \cup M_u^R$,

$$f_{\max}(\pi_v) \geqslant f_{\max}(\pi).$$

Proof of Lemma 2 is given in the Appendix.

In case of $f_{\max}$, if $u = 1$ then $|M^{\max}| = 0$. However when the first job of a permutation is critical, we can make use of the following result.

Lemma 3

In the problem with objective function $f_{\max}$, if the critical job $\pi(u)$ of a permutation π satisfies the condition $u = 1$, then π is an optimal permutation.

Proof

This is obvious, since for every j, $f_j(S_{g(j)} + p_j)$ is a lower bound on the minimum of $f_{\max}$, and simultaneously, $f_{\max}(\pi) = f_{\pi(1)}(S_{g(\pi(1))} + p_{\pi(1)})$.

Finally we discuss the question of whether the neighbourhoods generated by $M^{\max}$ and M^{sum} possess the connectivity property. A neighbourhood determined by a set of moves U has the connectivity property if for any $\pi^1 \in \Pi$ there exists a finite sequence $\pi^1, \pi^2, \ldots, \pi^r$ such that $\pi^{i+1} \in \{\pi_v^i : v \in U\}$ for $i = 1, \ldots, r-1$, and π^r is an optimal permutation. This property guarantees only the possibility of finding an optimal solution, without assurance that it will be found. Equivalently, if this property does not hold, we are sure that there exists at least one instance and at least one initial permutation for which an optimal permutation cannot be found.

Since using the moves from M^{sum} and starting from an arbitrary permutation one can construct any other permutation in a finite number of moves, it is clear that the connectivity property is satisfied when the set of moves is defined by M^{sum}. In the case of $M^{\max}$ we have the following result.

Lemma 4

In the problem with objective function $f_{\max}$, the neighbourhood defined by the set of moves $M^{\max}$ satisfies the connectivity property.

Proof of Lemma 4 is given in the Appendix.

Tabu list and tabu status of a move

The tabu list is defined as a finite set T of ordered pairs of jobs. If move (x, y) is performed on permutation π, then we add to T the pair of jobs $(\pi(x), \pi(x+1))$ if $x < y$, or the pair $(\pi(x-1), \pi(x))$ if $x > y$; initially T is empty. If $|T| = LengthT$, where $LengthT$ is the tabu list size, before adding a new element to T, one must remove the oldest one. With respect to a permutation π, move (x, y) with $x < y$ has status tabu if at least one pair $(\pi(i), \pi(x))$ with $x + 1 \leqslant i \leqslant y$ belongs to T, and move (x, y) with $x > y$ has status tabu if at least one pair $(\pi(x), \pi(i))$ with $y \leqslant i \leqslant x - 1$ is in T.

It can easily be verified that when one uses the inverse permutation of π, $\pi_{INV}(\pi(i)) := i$, $i = 1, \ldots, n$, then straightforward testing the tabu status of m moves requires $O(\max\{n, m \cdot LengthT\})$ time. This computational requirement can be reduced in the following manner. Given a permutation π and a tabu list T, let $r(i) = \min\{j : j > i, (\pi(j), \pi(i)) \in T\}$, and $l(i) = \max\{j : j < i, (\pi(i), \pi(j)) \in T\}$ for $i = 1, \ldots, n$. Having the matrices $\{r(i)\}$ and $\{l(i)\}$, we test whether a move (x, y) is tabu or not in the following manner: for $x < y$, move (x, y) is tabu iff $r(x) \leqslant y$, and for $x > y$, move (x, y) is tabu iff $y \leqslant l(x)$.

In order to find the above matrices one can first set initial values $r(i) = \infty$, $l(i) = 0$, and find π_{INV} (in $O(n)$ time). Then, $\{r(i)\}$ and $\{l(i)\}$ can be recurrently computed in the following way: for $(a, b) \in T$, if $\pi_{INV}(b) < \pi_{INV}(a)$, then set $r(\pi_{INV}(b)) := \min\{r(\pi_{INV}((b)), \pi_{INV}(a)\}$ and $l(\pi_{INV}(a)) := \max\{l(\pi_{INV}(a)), \pi_{INV}(b)\}$; this requires $O(LengthT)$ time. The overall complexity of computing $\{r(i)\}$ and $\{l(i)\}$ is $O(\max\{n, LengthT\})$. Since testing whether a single move is tabu requires $O(1)$ time, the overall time requirement for testing m moves is $O(\max\{n, LengthT, m\})$. The reduction of computational complexity is evident since due to the search strategy described below, $m \approx 2n$.

Search strategy

Given a permutation π, the neighborhood of π is searched in the following manner. First, for each subset $U \in \{M_3^L, \ldots, M_u^L, M_1^R, \ldots, M_{u-1}^R\}$, for $f_{\max}$, or each $U \in \{M_3^L, \ldots, M_n^L, M_1^R, \ldots, M_{n-1}^R\}$, for f_{sum}, the best move in U, called a representative of U and denoted by $v(U)$, is chosen,

$$f(\pi_{v(U)}) = \min_{v \in U} f(\pi_v).$$

Denote by SR the set of representatives associated with permutation π. Thus $SR = \{v(U): U \in \{M_3^L, \ldots, M_u^L, M_1^R, \ldots, M_u^R\}\}$ for $f_{\max}$, and $SR = \{v(U): U \in \{M_3^L, \ldots, M_n^L, M_1^R, \ldots, M_{n-1}^R\}\}$ for f_{sum}. Observe that $|SR| = \max\{2u - 3, 0\}$ for $f_{\max}$, and $|SR| = 2n - 3$ for f_{sum}.

A representative $v \in SR$ is either allowed, if it does not have status tabu, or forbidden but profitable, if it has status tabu and satisfies the condition $f(\pi_v) < A(\pi)$, or forbidden and unprofitable, if it has status tabu and satisfies the condition $f(\pi_v) \geqslant A(\pi)$. Here, A is an aspiration function defined for π as the lowest value of the objective function obtained in the search process up to the time π is generated, including π. The move to be performed, called the best move, is selected as the one that amongst all allowed or forbidden but profitable representatives gives the lowest value of the objective function.

If all the representatives are forbidden and nonprofitable, the oldest element of the tabu list T is deleted and the search is repeated until an allowed representative is found.

Fast computing an objective function for all neighbours

Computing an objective function for all neighbours in order to find all representatives is the most time consuming part of a single iteration, about 95%, in our tabu search algorithm. Therefore reduction of this computational burden is the crucial question. Observe that computing the value of f for a single permutation requires $O(n)$ time, assuming that finding $f_j(C_j)$ requires $O(1)$ time; this assumption remains valid in the sequel. Since both considered neighbourhoods have in the worst case $(n-1)^2$ elements, straightforward computing the objective function for the entire neighbourhoods requires $O(n^3)$ time. In what follows we show that this can be done in $O(n^2B^2)$ time, and if all major setup times are equal, this computational complexity is $O(n^2)$. Since in practical situations $B \ll n$, the reduction of computational complexity is evident.

Given a permutation π and position x, let β be the permutation obtained from π by deleting the element $\pi(x) \equiv j$. Thus

$$\beta = (\beta(1), \ldots, \beta(n-1)) = (\pi(1), \ldots, \pi(x-1), \pi(x+1), \ldots, \pi(n)).$$

Let $\beta^y = (\beta(1), \ldots, \beta(y-1), j, \beta(y), \ldots, \beta(n-1))$ be the permutation obtained from β by shifting the elements $\beta(y), \ldots, \beta(n-1)$ by one position to the right and then putting the element j in position y. Clearly, $\beta^y = \pi_v$, where $v = (x, y)$. As a result, move $v = (x, y)$ with respect to π can be interpreted as adding the new element j to β on position y.

Let C_k and C_k^y be the earliest completion times of job k in β and β^y, respectively. The basic observation that stands behind the fast computing method is that the completion times of jobs $\beta(1), \ldots, \beta(y-1)$ are the same in both β and β^y and the completion times in β and β^y of each remaining job differ from each other by an amount from the set D, see Lemma 1, whose cardinality depends on B and does not depend on n. This allows us to compute $f(\beta^y)$ for all moves (x, y) from the set M_x on the basis of the completion times C_k computed *a priori* for permutation β.

Clearly,

$$C^y_{\beta(i)} = C_{\beta(i)} \quad \text{for } i = 1, \ldots, y-1,$$

$$C^y_j = C_{\beta(y-1)} + s_{\beta(y-1),\,j} + p_j,$$

$$C^y_{\beta(i)} = C_{\beta(i)} - s_{\beta(y-1),\,\beta(y)} + s_{\beta(y-1),\,j} + s_{j,\,\beta(y)} + p_j \quad \text{for } i = y, \ldots, n-1,$$

where $\beta(0) = 0$, $C_0 = 0$. By Lemma 1, $-s_{\beta(y-1),\,\beta(y)} + s_{\beta(y-1),\,j} + s_{j,\,\beta(y)} = d_{\beta(y-1),\,j,\,\beta(y)} \in D$. Define for convenience the operation MOS: given a set of real numbers A, MOS $A = \max_{a \in A} a$ in the problem with $f = f_{\max}$, and MOS $A = \sum_{a \in A} a$, when $f = f_{\text{sum}}$. Then, for every y such that $(x, y) \in M_x, f(\beta^y)$ can be written as follows:

$$f(\beta^y) = \text{MOS}\{\text{MOS}_{1 \leqslant i \leqslant y-1} f_{\beta(i)}(C_{\beta(i)}),\ f_j(C_{\beta(y-1)} + s_{\beta(y-1),\,j} + p_j),$$

$$\text{MOS}_{y \leqslant i \leqslant n-1} f_{\beta(i)}(C_{\beta(i)} + d_{\beta(y-1),\,j,\,\beta(y)} + p_j)\}.$$

Assume D, see Lemma 1, has z different elements $\Delta_1, \ldots, \Delta_z$, and define for $k = 1, \ldots, n-1$,

$$L_k = \text{MOS}_{1 \leqslant i \leqslant k} f_{\beta(i)}(C_{\beta(i)}),$$

$$R_{k,\,l} = \text{MOS}_{k \leqslant i \leqslant n-1} f_{\beta(i)}(C_{\beta(i)} + \Delta_l + p_j), \quad l = 1, \ldots, z.$$

Additionally, set $L_0 = -\infty$, $R_{n,\,l} = -\infty$ for $f = f_{\max}$, and $L_0 = 0$, $R_{n,\,l} = 0$ for $f = f_{\text{sum}}$. Then

$$f(\beta^y) = \text{MOS}\{L_{y-1}, f_j(C_{\beta(y-1)} + s_{\beta(y-1),\,j} + p_j), R_{y,\,l^0}\},$$

where for $y < n$, $l^0 \in \{1, \ldots, z\}$ is such that $d_{\beta(y-1),\,j,\,\beta(y)} = \Delta_{l^0}$, and for $y = n$, l^0 is any number from $\{1, \ldots, z\}$. Computational complexities of finding the matrices $\{C_{\beta(i)}\}$, $\{L_k\}$ and $\{R_{k,\,l}\}$ are $O(n)$, $O(n)$ and $O(nz)$, respectively. Given matrices $\{C_{\beta(i)}\}$, $\{L_k\}$ and $\{R_{k,\,l}\}$, computing $f(\beta^y)$ for a single move (x, y) requires $O(z)$ time to find l^0, and then $O(1)$ time to determine $f(\beta^y)$. In consequence, the overall complexity of a procedure computing the values of $f(\beta^y)$ for all $n - \min\{2, x\}$ moves from M_x is $O(nz)$, and for all moves from M, $M \in \{M^{\max}, M^{\text{sum}}\}$, $O(n^2 z)$. By the definition of D, $z \leqslant B^2 + 2$, and if $S_b = S$ for all b, then $z = 5$. Even if the time needed to find l^0 is reduced by using a suitable data structure for matrix $\{\Delta_1, \ldots, \Delta_z\}$, this will not affect the overall complexity since the matrix $\{R_{k,\,l}\}$ cannot be computed in a time shorter than $O(nz)$. Observe also that one can avoid storing the entire matrix $\{R_{k,\,l}\}$ if the computing of $f(\beta^y)$ is performed in the order $y = n$, $y = n-1$, and so on, and row y of $\{R_{k,\,l}\}$, necessary to computing $f(\beta^y)$, is determined on the basis of the previously computed row $y + 1$.

Detailed description

In algorithm *TS* (given below) we use the list *LH* of length *LengthLH* which contains the history of last *LengthLH* improvements of the value of f. An element of *LH* is given in the form of the fourfold (π, *SR*, *T*, *CountRep*), where π is the permutation, *SR* and *T* are the set of representatives and the tabu list, respectively, associated with π, and *CountRep* is the number of representatives in *SR* which remain to be analysed in sequel. Each time the currently best value of the objective function f^{TS} is improved (Step 4), the better permutation π and the associated $SR\backslash\{v\}$, *T*, and *CountRep* = *MaxRep* − 1 are added to *LH* (Step 3), where v is the move in *SR* performed from π and *MaxRep* is the maximum number of representatives associated with π which are to be analysed. If *LH* is full, then before adding a new element to *LH*, the oldest one is removed. Each time an element is taken from *LH*, this is the currently newest element.

We assume that if *MaxIter* successive iterations without improving f^{TS} have been performed, then the newest element (π, *SR*, *T*, *CountRep*) of *LH* is taken (Step 6) and the search process resumes from it. After finding the best representative v in *SR* (Step 2), this element is again added to *LH* in the form (π, $SR\backslash\{v\}$, *T*, *CountRep* − 1) (Step 3), if *CountRep* > 1 and $SR\backslash\{v\} \neq \phi$. Consequently, every permutation stored in *LH* can be used as the starting point at most *MaxRep* times. The algorithm stops when list *LH* is empty.

Algorithm *TS*

Initialization

Set $\pi := \pi^{INI}$, where π^{INI} is an initial permutation, $\pi^{TS} := \pi, f^{TS} := f(\pi)$, $T := \phi$, $LH := \phi$, $Iter := 0$, $CountRep := MaxRep$, $Store := true$.

Step 1. Set $Iter := Iter + 1$. Using the fast method described in Section 4, find the set of representatives SR for π. If $SR = \phi$, then STOP (this can happen when $f = f_{\max}$; in this case $SR = \phi$ implies $u = 1$, and by Lemma 3, π^{TS} is an optimal permutation).

Step 2. Find the best representative $v \in SR$ in accordance with the search strategy described earlier, and determine the modified tabu list T', according to the method given earlier.

Step 3. If $Store = true$ and $SR\backslash\{v\}$ is not empty and $CountRep > 1$, then add to LH the fourfold $(\pi, SR\backslash\{v\}, T, CountRep - 1)$.

Step 4. Set $\pi := \pi_v$, $T := T'$, $Store := false$. If $f(\pi) < f^{TS}$, then set: $\pi^{TS} := \pi$, $f^{TS} := f(\pi)$, $Iter := 0$, $CountRep := MaxRep$, $Store := true$, and go to *Step 1*.

Step 5. If $Iter < MaxIter$, then go to *Step 1*.

Step 6. If $LH \neq 0$, then take from list LH the fourfold $(\pi, SR, T, CountRep)$ and set $Store := true$, $Iter := 1$, and go to *Step 2*; Otherwise STOP (the best solution found by the algorithm is stored as π^{TS} and $f^{TS} = f(\pi^{TS})$).

Algorithm *TS* has four tuning parameters *LengthT*, *LengthLH*, *MaxRep* and *MaxIter* that are to be chosen experimentally.

Computational complexity of a single iteration in *TS* is determined by Steps 1 and 2. By the considerations shown earlier, the complexity of Step 1 is $O(n^2 z)$. In Step 2, we have to choose the best representative amongst $|SR|$ ones, which according to the considerations shown earlier requires $O(\max\{n, LengthT, |SR|\})$ time. Since $|SR| \approx 2n$ and, practically, $LengthT \leqslant n$, the computational requirement of Step 2 is $O(n)$. As a result, computational complexity of a single iteration in *TS* is $O(n^2 z)$, where $z \leqslant B^2 + 2$, and if $S_b = S$ for all b, then $z = 5$.

COMPUTATIONAL RESULTS

Algorithm *TS* was coded in Pascal and run on PC 486 DX (16 MIPS). A test problem was given by the data: n, B, $I_1 = \{1, \ldots, |I_1|\}$, $I_2 = \{|I_1| + 1, \ldots, |I_1| + |I_2|\}$, $\ldots$, $I_B = \{(\sum_{i=1}^{B-1} |I_i|) + 1, \ldots, n\}$, S_b, $1 \leqslant b \leqslant B$, s, p_j, d_j, w_j, $1 \leqslant j \leqslant n$. The test problems were generated as follows.

All the test problems were divided into two groups: Group I with $B = 2, |I_1| = |I_2|, S_1 = S_2 = 40$, $s = 20$. and Group II with $B = 4$, $|I_1| = |I_2| = |I_3| = |I_4|$, $S_1 = 30$, $S_2 = 40$, $S_3 = 50$, $S_4 = 60$, $s = 20$. In each group, two objective functions: $\max_{j \in J} w_j(C_j - d_j)$ and $\sum_{j \in J} w_j \max\{C_j - d_j, 0\}$ were tested for $n \in \{40, 80, 120, 160, 200\}$. For each Group, objective function and for each n, 10 sets of integer numbers p_j, d_j, w_j, $1 \leqslant j \leqslant n$, (10 test problems) were generated from the uniform distribution: $p_j \in \{1, \ldots, 60\}$, $d_j \in \{1, \ldots, n * 30\}$, $w_j \in \{1, \ldots, 10\}$. The ranges of data are motivated by the real production process.

For both objective functions, the algorithm used the initial permutation $\pi^{INI} = (1, 2, \ldots, n)$. In view of the definition of s_{ij} and the assumed job numeration, this permutation generates minimum total setup time. In case of the total weighted tardiness, it appears to be the best one amongst heuristic permutations generated by a number of reasonable priority rules. As regards the maximum weighted lateness, slightly better initial permutations can be obtained by using a heuristic based on the well-known algorithm of Baker *et al.*[1] for solving the standard counterpart of our problem. The method consists in applying Baker's algorithm successively to the problems with relaxed setups, and with job processing times being enlarged in each iteration by certain, varying with iteration, quantity. Since starting from initial permutations generated by the above heuristic had not impact on the behaviour of *TS*, we decided to use the permutation $(1, \ldots, n)$ as a starting point. This initial permutation is also suited for comparisons with other possible heuristic approaches to our problem.

In the absence of other methods of solving the problem and due to lack of close lower bounds, we decided to measure the quality of obtained permutations by showing only how much an initial

distance from some (weak) lower bound to the initial value of the objective function was reduced by applying the *TS* algorithm, that is we computed for each problem instance the following performance index:

$$\rho = 100\%(f(\pi^{INI}) - f(\pi^{TS}))/(f(\pi^{INI}) - LB),$$

where π^{TS} is obtained by *TS*, π^{INI} is the initial permutation, and *LB* is the lower bound of the minimum value of f. In the total weighted tardiness test problems, we used simply $LB = 0$. In the maximum weighted lateness instances, where the objective function may have negative values, we used the lower bound defined by the following recursion. For the problem with job set J', where $J' \subset J$, let $C(J')$ be the smallest possible completion time of all jobs from J', and $f^*_{\max}(J')$, the minimum of $f_{\max}$. Then

$$f^*_{\max}(J') \geqslant \max\{f_l(C(J')),\ f^*_{\max}(J'\backslash\{l\})\},$$

where l satisfies the condition: $l \in J'$ and $f_l(C(J')) = \min_{j \in J'} f_j(C(J'))$. Putting initially $J' = J$ and computing the values $f_l(C(J'))$ backwards, each time deleting job l from the current set J', we get a sequence $f_{l_1}(C(J'_1)), \ldots, f_{l_n}(C(J'_n))$, where $|J'_n| = 1$, whose maximum is the desired lower bound *LB*. Clearly, $C(J') = \sum_{i \in J'} p_j + \sum_{b \in Z(J')} S_b$, where $Z(J')$ is the set of family indexes, each of them having at least one job in J', $Z(J') = \bigcup_{j \in J'} \{g(j)\}$.

Algorithm *TS* was run with the following values of tuning parameters. In both problems it was assumed that $LengthT = 8$ and $LengthLH = 3$. In the maximum weighted lateness problem we set $MaxRep = 5$ and $MaxIter = 200$. In the total weighted tardiness one, $MaxRep = 4$ and $MaxIter$ was taken from the set $\{100, 200\}$ in accordance with the following rule: $MaxIter = 200$ when the value of f^{TS} has been improved (Step 4), and $MaxIter = 100$ after return to the previously best solution stored in *LH* (Step 5) (initially $MaxIter = 200$). The particular values of tuning parameters were chosen experimentally, as a result of a compromise between running time and solution quality.

The results of computational experiments are shown in Tables 1 and 2. Besides ρ, we also show the total number of performed iterations (*Iterations Total*), the number of an iteration in which the best permutation generated by the algorithm was found (*Iterations Best*), CPU time per iteration

TABLE 1. *Computational results for instances from Group I**

n	$\max_{1 \leqslant j \leqslant n}(C_j - d_j)$					$\sum_{j=1}^{n} w_j \max\{0, C_j - d_j\}$				
	ρ	Iterations		CPU [sec]		ρ	Iterations		CPU [sec]	
		Total	Best	Iter	Best		Total	Best	Iter	Best
40	91.6	2982	480	0.01	5	74.6	1296	196	0.04	8
80	95.1	3803	1473	0.05	74	80.2	1193	93	0.15	14
120	93.6	2809	1168	0.11	128	82.4	1256	156	0.34	53
160	92.0	3600	998	0.17	170	83.1	1305	205	0.61	125
200	93.0	4353	1832	0.28	513	83.2	1490	390	0.95	370
all	93.1	3509	1190			80.7	1308	208		

* Each entry represents an average obtained from 10 instances of the given size.

TABLE 2. *Computational results for instances from Group II**

n	$\max_{1 \leqslant j \leqslant n}(C_j - d_j)$					$\sum_{j=1}^{n} w_j \max\{0, C_j - d_j\}$				
	ρ	Iterations		CPU [sec]		ρ	Iterations		CPU [sec]	
		Total	Best	Iter	Best		Total	Best	Iter	Best
40	91.5	3207	766	0.02	15	59.3	1198	98	0.05	5
80	91.8	3440	919	0.06	55	67.1	1412	312	0.22	69
120	91.8	4257	1776	0.13	231	68.9	1208	108	0.50	54
160	89.1	4673	2392	0.22	526	77.9	1374	274	0.89	244
200	90.8	4724	2163	0.34	735	79.8	1365	265	1.38	366
all	91.0	4060	1603			70.6	1311	211		

* Each entry represents an average obtained from 10 instances of the given size.

(*CPU Iter*), and CPU time by which the best permutation was found (*CPU Best*). For both objective functions, we observed significant reductions of the initial values $f(\pi^{INI})$. For maximum weighted lateness, average ρ in Group I was 93%, and in Group II, 91%, while for the total weighted tardiness, it was 81% and 71%, respectively. The above reductions were obtained after a few seconds, in case of 40 jobs, and in 6–12 minutes, for 200 jobs. This is due to the fact that in almost all instances about 80% of the total reduction was obtained in the first subsequent iterations.

Returns to the previously best solutions stored in *LH* appeared to have an impact on the output permutations mostly in the case of maximum weighted lateness. Improvements were observed after returns to all three levels of *LH*. However, from 95% to 98% of the decreasing of $f(\pi^{INI})$ had been obtained before returns commenced.

CONCLUSIONS

We have presented a tabu search algorithm that is capable of handling any single machine scheduling problem with assumed structure of major and minor setup times and a general regular cost function. Although we cannot show how the solutions obtained differ from the optimal ones, the computational results based on the assumed performance index (which is the relative reduction of the initial value of the objective function) suggest that our problem may be solved efficiently by a tabu search technique on personal computers. Every cost function requires experimental choice of the tuning parameters, but the method is easily implemented and is easily tuned; values of this parameters for tested cost functions are given.

The results obtained encourage to extend the proposed tabu search technique to other hard problems of sequencing with setups and batching.

APPENDIX

Proof of Lemma 1

For a pair of jobs l, m, let $\delta_{lm} = 1$ if $g(l) = g(m)$, and $\delta_{lm} = 0$, otherwise. For an integer h, let $\mathbf{1}(h) = 1$, if $h > 0$, and $\mathbf{1}(h) = 0$, otherwise. Using this notation, one can express the setup time s_{lm} by $s_{lm} = s\mathbf{1}(l - m)\delta_{lm} + S_{g(m)}(1 - \delta_{lm})$ which allows us to write d_{ijk} as follows

$$d_{ijk} = s[\mathbf{1}(i - j)\delta_{ij} + \mathbf{1}(j - k)\delta_{jk} - \mathbf{1}(i - k)\delta_{ik}] + S_{g(j)}[1 - \delta_{ij}] + S_{g(k)}[\delta_{ik} - \delta_{jk}].$$

Consider separately the following exhaustive cases.

Case: $\delta_{ij} = 0$, $\delta_{jk} = 0$. Then $d_{ijk} = -s\mathbf{1}(i - k)\delta_{ik} + S_{g(j)} + S_{g(k)}\delta_{ik}$. If $\delta_{ik} = 0$, then $d_{ijk} = S_{g(j)}$; otherwise, $d_{ijk} = -s\mathbf{1}(i - k) + S_{g(j)} + S_{g(k)}$. Thus in both cases $d_{ijk} \in D$.

Case: $\delta_{ij} = 0$, $\delta_{jk} = 1$. Then $\delta_{ik} = 0$ and $d_{ijk} = s\mathbf{1}(j - k) \in D$.

Case: $\delta_{ij} = 1$, $\delta_{jk} = 0$. Then $\delta_{ik} = 0$ and $d_{ijk} = s\mathbf{1}(i - j) \in D$.

Case: $\delta_{ij} = 1$, $\delta_{jk} = 1$. Then $\delta_{ik} = 1$ and $d_{ijk} = s[\mathbf{1}(i - j) + \mathbf{1}(j - k) - \mathbf{1}(i - k)]$. It can easily be verified that $\mathbf{1}(i - j) + \mathbf{1}(j - k) - \mathbf{1}(i - k)$ takes values 0 or 1, hence $d_{ijk} \in D$.

Proof of Lemma 2

First, consider a move $v = (x, y) \in \bigcup_{j=u+1}^{n} M_j$. It consists in removing a job that does not belong to a critical sequence of π from its position and inserting it in any other position. Denote by $C'_{\pi(u)}$ the earliest completion time of $\pi(u)$ in permutation π_v. It is clear from the definition of a critical job that if the position in which the job is inserted is greater than u, then $C'_{\pi(u)} = C_{\pi(u)}$ and $f_{\max}(\pi_v) \geqslant f_{\max}(\pi)$. If the job is inserted in the position less or equal to u, that is when $y \leqslant u < x$, then we have

$$C'_{\pi(u)} = C_{\pi(u)} - s_{\pi(y-1),\,\pi(y)} + s_{\pi(y-1),\,\pi(x)} + s_{\pi(x),\,\pi(y)} + p_{\pi(x)} \geqslant C_{\pi(u)},$$

where the inequality follows from the fact that s_{ij} satisfy the triangle inequality. This implies $f_{\max}(\pi_v) \geqslant f_{\max}(\pi)$.

Suppose now that $v = (x, y) \in M_u^R$. This move consists in removing $\pi(u)$ from position u and inserting it in the position y with $y > u$. In this case, by the triangle inequality and the fact that $p_j > 0$,

$$\begin{aligned} C'_{\pi(u)} &= C_{\pi(u)} - p_{\pi(u)} - s_{\pi(u-1),\,\pi(u)} + s_{\pi(u-1),\,\pi(u+1)} + p_{\pi(u+1)} \\ &\quad + \textstyle\sum_{i=u+2}^{y} (s_{\pi(i-1),\,\pi(i)} + p_{\pi(i)}) + s_{\pi(y),\,\pi(u)} + p_{\pi(u)} \\ &\geqslant C_{\pi(u)} - s_{\pi(u-1),\,\pi(u)} + s_{\pi(u-1),\,\pi(y)} + s_{\pi(y),\,\pi(u)} \geqslant C_{\pi(u)}. \end{aligned}$$

Consequently, $f_{\max}(\pi_v) \geqslant f_{\max}(\pi)$.

Proof of Lemma 4

For a permutation π, denote by $Y(\pi)$ the set of all order pairs of jobs consistent with π, $Y(\pi) = \{(\pi(i), \pi(j))\colon 0 \leqslant i < j \leqslant n\}$; recall that $\pi(0) \equiv 0$. For π and π', let $\rho(\pi, \pi') = |\, Y(\pi) \backslash Y(\pi') \,|$ define a distance from π to π'. Observe that if $\rho(\pi, \pi') = 0$, then $\pi = \pi'$. Under this notation, in order to prove the lemma, it suffices to show that if for some π with the set of moves $M^{\max}$, $f_{\max}(\pi) > f_{\max}(\pi^*)$, where π^* is an optimal permutation, then there exists move $v \in M^{\max}$ such that $\rho(\pi_v, \pi^*) < \rho(\pi, \pi^*)$. Since $\rho(\pi, \pi^*) \leqslant n(n+1)/2$ for any π, a finite number of such moves suffices to reach an optimal permutation. We now show that the above property holds.

Let π be an arbitrary permutation with critical job $\pi(u)$, and let $M^{\max}$ be the set of moves associated with π. Suppose that $f_{\max}(\pi) > f_{\max}(\pi^*)$, and consider the set of pairs $A = \{\pi(i), \pi(i+1))\colon i = 0, 1, \ldots, u-1\}$.

Let $A \subset Y(\pi^*)$. Then π^* has the form of $(\ldots, \pi(1), \ldots, \pi(2), \ldots, \pi(u), \ldots)$, and since s_{ij} satisfies the triangle inequality, the earliest completion time of job $\pi(u)$ in permutation π^*, denoted by $C^*_{\pi(u)}$, fulfils the inequality

$$C^*_{\pi(u)} \geqslant \sum_{k=1}^{u} (s_{\pi(k-1),\,\pi(k)} + p_{\pi(k)}) = C_{\pi(u)}.$$

This, however, implies $f_{\max}(\pi^*) \geqslant f_{\max}(\pi)$ which, together with $f_{\max}(\pi) > f_{\max}(\pi^*)$, yields a contradiction.

Thus $A \not\subset Y(\pi^*)$. This means that there exists a pair $(\pi(k), \pi(k+1)) \in A$ that does not belong to $Y(\pi^*)$ and such that $0 < k \leqslant u-1$. Obviously, the move $v = (k, k+1)$ belongs to $M^{\max}$. This move applied to π reverses the order of jobs $\pi(k)$ and $\pi(k+1)$, which in consequence yields $Y(\pi_v) = [Y(\pi) \backslash \{(\pi(k), \pi(k+1))\}] \cup \{(\pi(k+1), \pi(k))\}$. Clearly, $(\pi(k+1), \pi(k)) \in Y(\pi^*)$ and $(\pi(k), \pi(k+1)) \in Y(\pi)$. Hence

$$Y(\pi_v) \backslash Y(\pi^*) = [Y(\pi) \pi \backslash Y(\pi^*)] \backslash \{(\pi(k), \pi(k+1))\}.$$

and $\rho(\pi_v, \pi^*) = \rho(\pi, \pi^*) - 1 < \rho(\pi, \pi^*)$.

REFERENCES

1. C. N. Potts and L. N. Van Wassenhove (1992) Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *J. Opl Res. Soc.* **43**, 395–406.
2. C. L. Monma and C. N. Potts (1992) Analysis of heuristic for preemptive parallel machine scheduling with batch setup times. *Opns. Res.* **41**, 981–993.
3. S. Zdrzałka (1991) Approximation algorithms for single machine sequencing with delivery times and unit batch setup times. *Eur. J. Opl Res.* **51**, 199–209.
4. S. Zdrzałka (1994) Preemptive scheduling with release dates, delivery times and sequence independent setup times. *Eur. J. Opl Res.* **76**, 60–71.
5. F. Glover (1989) Tabu search: Part I. *ORSA J. Computing* **1**, 190–206.
6. F. Glover (1990) Tabu search: Part II. *ORSA J. Computing* **2**, 4–32.
7. R. J. M. Vaessens, E. H. L. Aarts and J. K. Lenstra (1994) Job shop scheduling by local search. Memorandum COSOR 94-05, Eindhoven University of Technology, Eindhoven.
8. F. Glover and R. Hubscher (1994) Applying tabu search with influential diversification to multiprocessor scheduling. *Comps and Opns Res.* **21**, 877–884.

9. E. NOWICKI and C. SMUTNICKI (1996) A fast tabu search algorithm for the permutation flow shop problem. *Eur. J. Opl Res.* **91**, 160–175.
10. E. NOWICKI and C. SMUTNICKI (1996) A fast taboo search algorithm for the job shop problem. *Mgmt Sci.* **42**, 797–813.
11. J.BRUNO and P. DOWNEY (1978) Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM J. Comput.* **7**, 393–404.
12. K. R. BAKER, E. L. LAWLER, J. K. LENSTRA and A. H. G. RINNOOY KAN (1983) Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Opns. Res.* **31**, 381–386.
13. J. K. LENSTRA, A. H. G. RINNOOY KAN and P. BRUCKER (1977) Complexity of machine scheduling problems. *Ann. Discrete Math.* **1**, 343–362.

Received January 1995; accepted February 1996 after one revision