ELSEVIER
European Journal of Operational Research 109 (1998) 142–159
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

An adaptive TS approach to JIT sequencing with variable processing times and sequence-dependent setups

F. Kolahan, M. Liang [*]

Department of Mechanical Engineering, Faculty of Engineering, University of Ottawa, P.O. Box 450, Ottawa, Ontario, Canada K1N 6N5

Received 4 April 1996; accepted 13 January 1997

Abstract

This paper addresses a single machine sequencing problem with variable processing times and sequence-dependent setups. The objective is to find the best trade-off between the JIT goal and the processing time compression and extension costs by simultaneously determining the job sequence and processing times for concerned jobs. Due to the combinatorial nature of the problem, it cannot be optimally solved in polynomial time. A tabu search approach is used to provide good and quick solutions. To improve the computational efficiency, an adaptive neighbourhood generation method is proposed and used in the tabu search algorithm. A total of 100 problems of different sizes have been solved to test the proposed approach. Our computational experience shows that the adaptive approach outperforms several other neighbourhood generation methods in terms of both convergence rate and solution quality. The effects of the search parameters are also discussed. © 1998 Elsevier Science B.V.

Keywords: Tabu search; Adaptive neighbourhood generation; Just-in-time production; Job sequencing; Variable processing time

1. Introduction

Simultaneous consideration of both earliness and tardiness in scheduling decisions has been motivated by the Just-In-Time (JIT) production philosophy over the past decades. Excellent survey and research work has been carried out by several researchers (e.g., [13,2,20,8,17]).

Numerical control technology has been widely used to improve process productivity and controllability. However, the good controllability of machining centers in adjusting processing times has not been fully utilized to assist in achieving the JIT goal. This is often reflected in the conventional machine sequencing practice. With conventional sequencing methods, the processing time of each job is considered to be pre-determined and the machine-level decisions have little influence on the job-level decisions such as processing time specifications. Although several researchers (e.g., [21,6,16,3]) have initiated the studies on the sequencing problem with controllable processing times during the past few years, most of the them consider only time compression and do not address JIT sequencing goal, i.e., minimization of both tardiness and earliness costs. Moreover, the effect of sequence-dependent setup times has often been ignored and a common due date is usually assumed for all jobs.

[*] Corresponding author. Fax: +1-613-5625177.

PII S0377-2217(97)00098-2

In summary, most of the previous studies either consider JIT objective while overlooking the time controllability or address time controllability (mostly only compression) without the JIT objective. The sequence-dependent setup time is usually not considered. However, the reality in many shop floors equipped with machining centers often features: (a) both earliness and tardiness are undesirable; (b) the processing time of a job can be both compressed and extended by adjusting feed rate, depth of cut and spindle speed at extra cost; and (c) the setup time is usually sequence-dependent and each job may have its own distinct due date. There is thus a great need to further explore the sequencing problems in the modern shop floor.

The JIT sequencing problem with variable processing times and sequence-dependent setups will be investigated in this paper. Due to the combinatorial nature of the problem, an adaptive tabu search algorithm is proposed to provide quick solutions. We introduce a new neighbourhood generation mechanism to reduce the computational time. In the propose method, only the most promising neighbours are generated for each move. The details of the problem and solution procedure are presented in the following sections.

2. Problem statement

Consider J jobs to be processed on a machining center. Each job has its own due date. Any earliness or tardiness will lead to penalty cost and thus is undesirable. The setup time for each job is sequence-dependent as the setup time involved in changing tools and part holders depends, to some extent, on the similarity in tool and fixture requirements between the current job and the immediate previous one. The processing time of each job is controllable within a feasible range by adjusting feed rate, spindle speed and/or depth of cut. However, these adjustments will involve certain costs. The cost components to be considered in this paper are described below.

2.1. Tardiness and earliness penalties

In the JIT environment, if a job is completed before its due date, an earliness penalty will be considered because of the increased inventory, cash commitment, and possible shop floor congestion. The tardiness penalty usually occurs due to the loss of goodwill if the job is to be delivered to the customer or due to the waiting time if the job is to be processed by the next manufacturing stage. The amount of earliness or tardiness of a job depends on the processing and setup times of the concerned job and those of all the other jobs processed prior to it.

2.2. Compression and extension costs

Usually, in a machining process the processing cost for a job consists of operating cost, tool cost and defective cost. The operating cost includes labour cost and machine overhead, and increases with time. The defective cost is mainly caused by severe tool wear and in-process tool breakages. When the process is sped up, the tool and defective costs increase due to the increased tool wear and in-process tool failure. There is a best combination of machining parameters (feed rate, spindle speed and depth of cut) which minimizes the total processing cost of a job. In this paper, we define the processing time associated with such a best combination of machining parameters as the *normal processing time*. Any deviation from the normal processing time will cause extra cost. The *additional* cost due to process speedup is defined as the *compression cost* and that caused by process slowdown is called *extension cost*.

Although the normal processing time is preferred for a single job, it may not necessarily be beneficial at the machine-level when all jobs' due dates are taken into account. It may be desirable to choose processing times other than the normal processing times for some jobs so that the sum of the earliness and tardiness penalty, and compression and extension costs is minimized at machine-level. The processing time selection also has impact on the optimal job sequence. The optimal job sequence obtained based on a particular set of processing times may no longer be optimal when a different set of processing times is used. Therefore, our objective is to jointly solve the job sequencing and processing time selection problems so that the total earliness and tardiness penalties, as well as compression and extension costs can be minimized.

The sequencing problems with sequence-dependent setup times and fixed processing times are well known NP-hard problems [18]. The earliness-tardiness sequencing problems with fixed processing times are also NP-hard [10,4,19]. The problem under consideration is a more general one which involves both sequence-dependent setups and earliness-tardiness issues and is further complicated by the variable processing times. Hence, such a problem cannot be optimally solved in polynomial time.

3. Problem formulation

3.1. Notation

The following notations are used for problem formulation while those used in the tabu search algorithm will be explained therein.

i,j	job index, $i,j = 1,\dots,J$,
l,k	position index, $l = 1,\dots,J$,
J	total number of jobs,
s	part sequence index, denoting a specific permutation of J jobs,
$\mu(s,l)$	the job in lth position of sequence s,
t_{ij}	setup time required by job j if it is processed immediately after job i,
t_{0j}	setup time of job j if it is the first job being processed in the sequence,
d_j	due date of job j,
T_j	tardiness of job j,
E_j	earliness of job j,
a_j	penalty cost per unit time tardiness for job j,
b_j	penalty cost per unit time earliness for job j,
P_j	current processing time of job j,
P_j^{N}	normal processing time of job j,
P_j^{L}	minimum possible processing time of job j,
P_j^{U}	maximum allowed processing time of job j,
X_j	amount of compressed processing time of job j; $0 \le X_j \le P_j^{\mathrm{N}} - P_j^{\mathrm{L}}$,
Y_j	amount of extended processing time of job j; $0 \le Y_j \le P_j^{\mathrm{U}} - P_j^{\mathrm{N}}$,
α_j	cost of reducing processing time of job j by one unit (compression cost),
β_j	cost of increasing processing time of job j by one unit (extension cost),
F_j^n	cost saving achieved by increasing processing time of job j by one unit in iteration n,
V_j^n	maximum allowed time increase for job j in iteration n; $0 \le V_j^n \le P_j^{\mathrm{U}} - P_j^{\mathrm{L}}$,
$G(s)$	total cost associated with sequence s,
W_s	a 0-1 integer variable, $W_s = 1$, if sequence s is selected; 0, otherwise,
$N(s)$	set of job sequences in the neighbourhood of s,
$N_{\mathrm{a}}(s)$	set of job sequences in the adaptively generated neighbourhood of s.

3.2. The model

The JIT sequencing problem with variable processing times and sequence-dependent setup times can be formulated as the following nonlinear mixed integer programming model:

Model 1

$$\operatorname*{Min}_{1\le s\le J!}\{G(s)\} = \text{Min} \sum_{s=1}^{J!} W_s \sum_{l=1}^{J} \left(a_{\mu(s,l)}T_{\mu(s,l)} + b_{\mu(s,l)}E_{\mu(s,l)} + \alpha_{\mu(s,l)}X_{\mu(s,l)} + \beta_{\mu(s,l)}Y_{\mu(s,l)}\right) \tag{1}$$

subject to:

$$P_{\mu(s,l)} + X_{\mu(s,l)} \ge P^{\mathrm{N}}_{\mu(s,l)}, \quad \forall s,l, \tag{2}$$

$$P_{\mu(s,l)} - Y_{\mu(s,l)} \le P^{\mathrm{N}}_{\mu(s,l)}, \quad \forall s,l, \tag{3}$$

$$P^{\mathrm{L}}_{\mu(s,l)} \le P_{\mu(s,l)} \le P^{\mathrm{U}}_{\mu(s,l)}, \quad \forall s,l, \tag{4}$$

$$\sum_{k=1}^{l} \left(t_{\mu(s,k-1),\mu(s,k)} + P_{\mu(s,k)}\right) - d_{\mu(s,l)} \le T_{\mu(s,l)}, \quad \forall s,l, \tag{5}$$

$$d_{\mu(s,l)} - \sum_{k=1}^{l} \left(t_{\mu(s,k-1),\mu(s,k)} + P_{\mu(s,k)}\right) \le E_{\mu(s,l)}, \quad \forall s,l \tag{6}$$

$$E_{\mu(s,l)} \ge 0, \quad \forall s,l, \tag{7}$$

$$T_{\mu(s,l)} \ge 0, \quad \forall s,l, \tag{8}$$

$$\sum_{s=1}^{J!} W_s = 1, \tag{9}$$

where $t_{\mu(s,k-1),\mu(s,k)} = t_{0k}$ when $k = 1$.

The objective function contains four terms. The first two terms are respectively associated with tardiness and earliness while the last two are related to compression and extension costs. The objective function, in conjunction with constraint (9), states that only the sequence which minimizes the four cost components should be selected. Constraints (2) and (3) specify the relationship between the actual processing time and the normal processing time. The range of the actual processing time is given by constraint (4). Constraints (5) and (6) define the relationship between a job's completion time, due date, tardiness and earliness. The completion time includes cumulative setup and processing times. It should be pointed out that the setup time of a job is not included in the processing time since it is sequence-dependent. It is also noted that the tardiness and earliness of a job cannot co-exist. This is ensured by jointly considering constraints (5) to (8). This model cannot be easily solved even for small size problems because of the non-linearity of the objective function and the large number of 0-1 variables, W_s. To provide an efficient solution procedure for this problem, we propose an adaptive tabu search approach. The details of the proposed technique are presented in the following section.

4. Solution procedure

4.1. Tabu search algorithm

Tabu search, proposed in its present form by Glover [11], is a useful search technique for solving large combinatorial optimization problems. With this method, the search starts with a feasible solution and moves stepwise to a neighbouring solution in an attempt to obtain an optimal or near-optimal solution. The distinct characteristic of tabu search lies in its ability to escape from local optima by accepting non-improving solutions during the search process. Another important feature of tabu search is the use of tabu list for short term memory. A tabu list contains a number of immediate previous moves which are not allowed at the current iteration. The use of tabu list alleviates the cycling problem since the search is prohibited from returning to any of the previous moves specified in the tabu list. Following each move, the tabu list is updated by adding the new move and removing the oldest move from the list. To enhance the search performance, some path diversification strategies can also be applied. The details of tabu search and its refined versions are well documented in the literature (e.g. [12,14,15,23]).

4.2. Neighbourhood generation and move selection

Conventionally, a neighbourhood, $N(s)$, is defined as a set of solutions that can be obtained by performing one transition in the current solution. In the scheduling literature, *pairwise interchange* is probably the most widely used technique to make such a transition. In this method, a solution is generated by switching the jobs in positions i and j. The complete pairwise interchanges of a J-job problem leads to $|N(s)| = J(J-1)/2$ neighbours. *Extraction and reinsertion* is another technique for transition. With this method, the neighbourhood of s, $N_I(s)$, contains all solutions obtained by extracting the job in position i and inserting it right after (or before) the job in position j. The neighbourhood size for the extraction and reinsertion approach is $|N_I(s)| = J(J-1)^2$ which is almost doubled as compared to pairwise interchange and thus this mechanism appears to be more computationally demanding. Furthermore, as indicated by Adenso-Diaz [1], none of these two techniques seems to outperform the other in terms of solution quality for a given run time. Therefore, in this paper only the pairwise interchange approach is used as the basic neighbourhood generation mechanism.

The next step in tabu search is to specify a move strategy. The classic approach is to evaluate the entire neighbourhood and choose the best allowable move. However, the required computational time could be unacceptably long when the problem size is large. To overcome this problem, partial search schemes have been proposed recently. For instance, Crauwel et al. [5] used the first non-tabu move that results in a superior solution, and William [22] and Della Croce [7] selected the best non-tabu move from a fixed number of randomly or sequentially generated solutions.

A more efficient and sophisticated strategy is probably to evaluate part of neighbours that most likely contain the best solution [14]. Along this line, Andeso-Diaz [1] proposes the use of a *restricted neighbourhood* (RN) to improve the computational performance in solving a weighted tardiness problem. This approach is based on the observation that the convergence process generally presents an exponential behaviour. It is further observed that, in the first few iterations, the largest cost reduction (objective improvement) is often obtained by exchanging the jobs located far apart in the current sequence. Thus, larger transposition ranges tend to be used in the first few iterations. As the search progresses, the transposition ranges become smaller. In view of this, Andeso-Diaz [1] proposes that the entire neighbourhood should be evaluated only in the first few iterations whereas the transposition ranges in the later iterations should be consecutively reduced to a pre-specified number of neighbouring jobs.

However, most of the above search processes are not guided by the objective value. For instance, in one of the best approaches [1], all the parameters used in determining the range of transpositions are pre-specified and are maintained throughout the entire search process. This approach cannot fully utilize the information readily available in the search process such as the objective value and its improvement rate right after each iteration. To this end, an adaptive approach is proposed in the next section to determine the range of transpositions in response to the search dynamics.

4.3. Proposed adaptive neighbourhood generation (ANG)

The main idea of the proposed ANG method is to maintain a large transposition range when a large objective improvement is observed and reduce the transposition range if the a small objective improvement or no improvement is observed. In contrast to the other tabu search methods, the ANG features the following two strategies: (a) objective-guided adaptive adjustment of transposition ranges; and (b) double-ended transposition range reduction. To begin with, the total number of iterations, *Mmax*, is specified and divided into equal intervals each with R iterations. For an interval, say r, the above strategies are implemented using the following two equations:

$$\mathrm{RA}(r) = \begin{cases} 1 & \text{if} \quad Z \le 1, \\ z & \text{if} \quad 1 < Z < J-1, \\ J-1 & \text{if} \quad Z \ge J-1, \end{cases} \tag{10}$$

$$\mathrm{ra}(r) = \begin{cases} 1 & \text{if} \quad z \le 1, \\ z & \text{if} \quad 1 < z < \mathrm{RA}(r), \\ \mathrm{RA}(r) & \text{if} \quad z \ge \mathrm{RA}(r), \end{cases} \tag{11}$$

where

$$Z = \left[\mathrm{RA}(r-1) - Y_{\mathrm{RA}} J^{1-(\lambda_{r-1}/\lambda_0 e^{-\Psi r})}\right],$$

$$z = \left[\mathrm{ra}(r-1) - Y_{\mathrm{ra}} J^{1-(\lambda_{r-1}/\lambda_0 e^{-\psi r})}\right],$$

λ_r is the maximum improvement obtained between the two successive iterations in interval r (%); λ_0 the aspired improvement for the first interval (%); ψ, Ψ are damping factors of the aspired improvement; Y_{RA} a 0-1 integer variable, $Y_{\mathrm{RA}} = 1$, if $\lambda_{r-1} < \lambda_0 e^{-\Psi r}$; 0, otherwise; Y_{ra} a 0-1 integer variable, $Y_{\mathrm{ra}} = 1$, if $\lambda_{r-1} < \lambda_0 e^{-\psi r}$; 0, otherwise.

RA(r) and ra(r) respectively define the upper and lower boundaries of the transposition range for interval r and they are adjusted adaptively based on an objective improvement indicator, i.e., the ratio λ_{r-1}/λ_0. The purpose of dividing the entire search process into intervals is to avoid the sudden drop of transposition range due to the possible non-improving moves. As the objective value in the later iterations usually does not improve as fast as it does in the first few iterations, the aspiration level should be reduced accordingly since otherwise the range will be reduced very quickly. For this purpose, the damping factors ψ and Ψ are used to restrain the reduction process of the transposition range. Using the above expressions, the neighbourhood size can be calculated as $|N_a(s)| = J[J - \mathrm{ra}(r)] - 1/2[J - \mathrm{ra}(r)][\mathrm{ra}(r) + \mathrm{RA}(r)]$ which is in the range of $[J+1, |N(s)|]$ (Note: $|N(s)| = J(J-1)/2$) and thus generally $|N_a(s)| \le |N(s)|$).

The proposed ANG method may be better illustrated using Fig. 1. For comparison, the typical patterns of the objective values (Fig. 1(a)) and the transposition ranges (Fig. 1(b)) obtained using both the ANG and RN methods are plotted. In Fig. 1(b), the area between the two solid curves is the transposition range corresponding to the ANG search process with RA as the upper boundary (end) and ra the lower boundary (end). The entire area under the dash-dot curve represents the transposition range obtained using the RN method. As shown in the figure, the ANG range is significantly narrower than the RN range. It can also be seen that the ANG transposition range in Fig. 1(b) is adaptively adjusted in response to the objective improvement rate (Fig. 1(a)) while the RN transposition range has no correspondence to its objective value.

4.4. Tabu search for JIT sequencing

In the JIT sequencing problem, a solution is essentially a permutation of J jobs. There is a total of $J!$ possible sequences or permutations in the entire solution domain. For the adaptively generated neighbourhood, each neighbour in $N_a(s)$ has its associated objective function value, $G(s)$, and the one with the lowest $G(s)$ is defined as the best neighbour, denoted as s^*. A move is then made from s^{**}, the best job sequence of the immediate previous neighbourhood, to s^*, provided that s^* is not in the current tabu list, *T_list*. The best solution found so far, *Cbest*, and its corresponding sequence, *S_best*, are then updated if necessary and kept in memory. The sequence s^{**} is then stacked into the list *T_list* of size *T_size* and the oldest sequence is removed from the list. In addition, for each interval, the maximum cost reduction rate, *λ_best*, is retained to

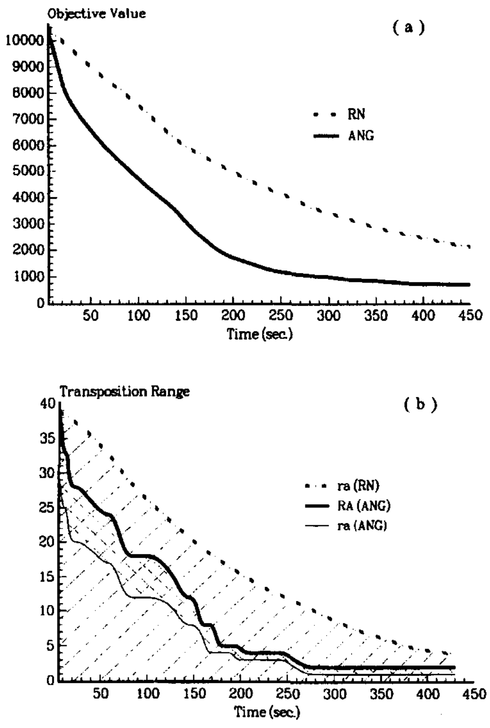

Fig. 1. Transposition range and objective value.

determine the transposition range for the next interval. This procedure is repeated until a specified termination criterion is reached. For the example problem in this paper, the maximum number of iterations, *Mmax*, is used as the termination criterion.

The tabu search algorithm for the JIT sequencing problem using the ANG method is presented below.

Algorithm 1

```
Step 1. Initialization.
(a) Set $T\_list = \{\emptyset\}$, $S\_best = \{\emptyset\}$, $M\_ctr$ (a counter) $= 0$, $r = 1$, $\lambda\_best = -\infty$, and $Cbest = \infty$
(b) Read $T\_size$, $Mmax$, $R$, $\lambda_0$, $\psi$, $\Psi$, ra(1), RA(1), and $P_j^L$, $P_j^N$, $P_j^U$, $\alpha_j$, $\beta_j$, $a_j$, $b_j$, $d_j$ for $j = 1, \ldots, J$
(c) Construct a starting sequence $s^{**}$ and compute $G(s^{**})$
Step 2. Search.
Step 2.1. Generate and evaluate neighbouring solutions.
WHILE $M\_ctr < Mmax$ DO
set $G(s^*) = \infty$;
DO $jj = 1$ to $J - \text{ra}(r)$
DO $kk = \text{ra}(r)$ to $\text{RA}(r)$
generate a new neighbour $s$ for $s^{**}$ by interchanging the parts in positions $jj$ and $kk$.
IF $s \in T\_list$, discard $s$, and continue;
ELSE compute $G(s)$
IF $G(s) < G(s^*)$, set $G(s^*) \leftarrow G(s)$, and $s^* \leftarrow s$;
ELSE discard $s$, and continue;
ENDIF
```

```
ENDIF
ENDDO
ENDDO
Step 2.2. Move.
IF $\lambda\_best < (G(s^{**})/G(s^{*})) - 1$, set $\lambda\_best \leftarrow (G(s^{**})/G(s^{*})) - 1$;
ELSE continue;
ENDIF
Set $G(s^{**}) \leftarrow G(s^{*})$, $s^{**} \leftarrow s^{*}$ and update $T\_list$;
IF $G(s^{**}) < Cbest$, set $Cbest \leftarrow G(s^{**})$, $S\_best \leftarrow s^{**}$, and $M\_ctr \leftarrow M\_ctr + 1$;
ELSE set $M\_ctr \leftarrow M\_ctr + 1$;
ENDIF
Step 2.3. Calculate range of transpositions.
IF $r < (M\_ctr/R)$, set $r \leftarrow r + 1$, $\lambda_{r-1} \leftarrow \lambda\_best$, $\lambda\_best = -\infty$, calculate new ra($r$) and RA($r$);
ELSE continue;
ENDIF
ENDWHILE
Stop
Step 3. Diversification.
Diversify the search using one or more of the following strategies:
(1) Divide the maximum number of moves, Mmax, into equal sized phases of Nmax moves and after each phase:
(a) restart from the best solution found so far, S_best; or
(b) restart from a randomly selected sequence.
(2) Change the search parameters: T_size, Nmax, $\lambda_0$, $\psi$, $\Psi$, ra(1), RA(1), R and repeat the search.
```

The purpose of step 3 in the above algorithm is to enhance the search performance and to find the best combination of search parameters.

To evaluate the solutions in $N_a(s)$ at each iteration, it is necessary to calculate the minimum cost $G(s)$ for each sequence. This involves solving a set of linear programming models. The linear programming model and its solution procedure are described in the following.

4.5. Single sequence cost minimization

When the problem involves a given neighbour sequence, say s, Model 1 reduces to the following linear programming problem:

Model 2

$$\text{Min} \sum_{l=1}^{J} \left(a_{\mu(s,l)} T_{\mu(s,l)} + b_{\mu(s,l)} E_{\mu(s,l)} + \alpha_{\mu(s,l)} X_{\mu(s,l)} + \beta_{\mu(s,l)} Y_{\mu(s,l)} \right) \tag{12}$$

subject to:

$$P_{\mu(s,l)} - Y_{\mu(s,l)} \le P^{\mathrm{N}}_{\mu(s,l)}, \quad \forall l, \tag{13}$$

$$P_{\mu(s,l)} + X_{\mu(s,l)} \ge P^{\mathrm{N}}_{\mu(s,l)}, \quad \forall l, \tag{14}$$

$$P^{\mathrm{L}}_{\mu(s,l)} \le P_{\mu(s,l)} \le P^{\mathrm{U}}_{\mu(s,l)}, \quad \forall l, \tag{15}$$

$$\sum_{k=1}^{l} \left(t_{\mu(s,k-1),\mu(s,k)} + P_{\mu(s,k)}\right) - d_{\mu(s,l)} \le T_{\mu(s,l)}, \quad \forall l, \tag{16}$$

$$d_{\mu(s,l)} - \sum_{k=1}^{l} \left(t_{\mu(s,k-1),\mu(s,k)} + P_{\mu(s,k)}\right) \le E_{\mu(s,l)}, \quad \forall l, \tag{17}$$

$$E_{\mu(s,l)} \ge 0, \quad \forall l, \tag{18}$$

$$T_{\mu(s,l)} \ge 0, \quad \forall l. \tag{19}$$

This model can be solved using commercial software such as LINDO. It takes about 2 sec to solve Model 2 for a 30-job problem using LINDO. However, since Model 2 is nested in Algorithm 1 and each move involves solving a large number of such problems, it is practically not acceptable to use commercial software to solve Model 2. To reduce the computational burden, an efficient algorithm for Model 2 is developed in the next section.

4.6. Solution procedure for Model 2

Here, an efficient heuristic algorithm is developed for solving Model 2. To simplify the computational process, we start with a permutation in which all jobs are assumed to be processed with minimum allowed processing times. As a result, the further compression of processing time of a job is impossible and only extension of processing time can be made. The *saving rate*, i.e., the total potential saving in iteration n due to one unit time increment for the job in position k of sequence s is given by:

$$F^n_{\mu(s,k)} = B_{\mu(s,k)} - A_{\mu(s,k)} + H_{\mu(s,k)}, \tag{20}$$

where

$$B_{\mu(s,k)} = \sum_{l=k}^{J} b_l, \quad \{l \mid E^n_l > 0\}, \tag{21}$$

$$A_{\mu(s,k)} = \sum_{l=k}^{J} a_k, \quad \{l \mid T^n_l \ge 0\} \tag{22}$$

and

$$H_{\mu(s,k)} = \begin{cases} \alpha_{\mu(s,k)} & \text{if} \quad V^n_{\mu(s,k)} > P^{\mathrm{U}}_{\mu(s,k)} - P^{\mathrm{N}}_{\mu(s,k)}, \\ -\beta_{\mu(s,k)} & \text{if} \quad V^n_{\mu(s,k)} \le P^{\mathrm{U}}_{\mu(s,k)} - P^{\mathrm{N}}_{\mu(s,k)}. \end{cases} \tag{23}$$

V^n_j = maximum allowed time increase for job j in iteration n; $0 \le V^n_j \le P^{\mathrm{U}}_j - P^{\mathrm{L}}_j$.

In Eq. (20), the first two terms respectively take care of the earliness and tardiness costs caused by one unit time increment. $B_{\mu(s,k)}$ is the sum of the reduced earliness penalty on all the early jobs and $A_{\mu(s,k)}$ is the total increased tardiness penalty due to one unit time increment. The sign of $B_{\mu(s,k)}$ is positive since any time increase for the early jobs will reduce the earliness penalty which means a positive saving. In contrast, for the tardy jobs, any time increment will increase the tardiness penalty and, therefore, the second term has a negative sign. The last term in Eq. (20) reflects the effect of one unit time increment on the compression and extension costs. As shown in Eq. (23), if the increasable processing time is greater than the difference of $P^{\mathrm{U}}_{\mu(s,k)}$ and $P^{\mathrm{N}}_{\mu(s,k)}$, i.e., the current processing time of job $\mu(s,k)$ is less than $P^{\mathrm{N}}_{\mu(s,k)}$, one unit of time increase will release the compression and thus the effect is a saving of $\alpha_{\mu(s,k)}$. Otherwise, if the amount of increasable time is less than the difference of $P^{\mathrm{U}}_{\mu(s,k)}$ and $P^{\mathrm{N}}_{\mu(s,k)}$, namely, the current processing time of job $\mu_{\mu(s,l)}$ is greater than $P^{\mathrm{N}}_{\mu(s,k)}$, further slowing down the process by one time unit will reduce the saving by $\beta_{\mu(s,k)}$.

The above saving rate can be used as a measure of the potential improvement achievable by changing the processing time of each job in the permutation. The heuristic algorithm for solving Model 2 is based on the saving rate measure and the algorithm is summarized below.

Algorithm 2

Step 1. Set $n \leftarrow 1$ and compute the tardiness or earliness of each job assuming all jobs are processed with the minimum possible time.

Step 2. Compute the saving rate, $F^n_{\mu(s,l)}$, for all jobs with $V^n_{\mu(s,l)} > 0$. If $F^n_{\mu(s,l)} \le 0$ or $V^n_{\mu(s,l)} = 0$ for all jobs, further cost reduction cannot be achieved and the computation is terminated. Otherwise, go to next step.

Step 3. Find $F^n_{\mu(s,q)} = \max\{F^n_{\mu(s,l)} \mid \forall l\}$. Increase processing time of $\mu(s,q)$ by

$$\Delta P^n_{\mu(s,q)} = \min\{\delta_1, \delta_2\}, \tag{24}$$

where

$$\delta_1 = \min\left\{E^n_{\mu(s,q)} > 0 \mid l \ge q\right\} \tag{25}$$

and

$$\delta_2 = \begin{cases} V^n_{\mu(s,q)} + P^{\mathrm{N}}_{\mu(s,q)} - P^{\mathrm{U}}_{\mu(s,q)} & \text{if} \quad V^n_{\mu(s,q)} > P^{\mathrm{U}}_{\mu(s,q)} - P^{\mathrm{N}}_{\mu(s,q)}, \\ V^n_{\mu(s,q)} & \text{if} \quad V^n_{\mu(s,q)} \le P^{\mathrm{U}}_{\mu(s,q)} - P^{\mathrm{N}}_{\mu(s,q)}. \end{cases} \tag{26}$$

Step 4. Set $n \leftarrow n + 1$. Update $V^n_{\mu(s,q)}$, $E^n_{\mu(s,l)}$, $T^n_{\mu(s,l)}$, for $l \ge q$, and go to step 2.

Step 3 in the above algorithm is based on the logic that the time increase for the job with the highest saving rate is likely to be most profitable. Thus the time increase should be made until such high profitability has been exhausted. The range of such high profitable time is defined in step 3 and proven below.

Theorem 1. (*Selection of Time Increment*) *For a job $\mu(s,l)$ with increasable processing time in iteration n, i.e., $V^n_{\mu(s,k)} > 0$,*

(*a*) *if $F^n_{\mu(s,k)}$ is positive, it will remain constant in the range of $[P^n_{\mu(s,k)}, P^n_{\mu(s,k)} + \Delta P^n_{\mu(s,k)}]$ and the saving will be a monotonic linear increasing function with slope $F^n_{\mu(s,k)}$ in that range;*

(*b*) *if processing time is further increased above the range of $[P^n_{\mu(s,k)}, P^n_{\mu(s,k)} + \Delta P^n_{\mu(s,k)}]$, $F^n_{\mu(s,k)}$ will decrease though it may remain positive.*

Proof. See Appendix A.

As shown in Theorem 1, the saving rate level of the current job will become lower beyond certain range. Thereafter, even if the saving rate is still positive, we will not further increase the processing time in the current iteration. The reason is that some other jobs may yield a higher saving rate than that of the current job. Therefore, a new iteration using a newly selected highest saving rate will be carried out.

The termination criterion is based on the following theorem:

Theorem 2. (*Termination Criterion*) *In Algorithm 2, further improvement cannot be achieved in the current and future iterations if the maximum saving rate, $F^n_{\mu(s,q)} = \max\{F^n_{\mu(s,l)} \mid l = 1, \ldots, J\} \le 0$.*

Proof. See Appendix A.

Algorithm 2 has been coded in C + + and tested using 1000 randomly generated problems of different sizes on a 486 PC with a 33 MHz processor and 4 MB of RAM. The results obtained using the algorithm are

Table 1

Problem size (no. of jobs)	No. of problems solved	Accuracy of solutions	Average computational time (sec)	
			LINDO	Algorithm 2
10	200	100%	0.62	0.012
20	200	100%	1.10	0.018
30	200	100%	1.83	0.031
40	200	100%	2.55	0.049
50	200	100%	3.46	0.062

compared with the optimal solutions obtained using LINDO package (Table 1). As summarized in Table 1, Algorithm 2 provided exact solutions for all the 1000 problems while the computational times are considerably reduced. This leads us to conjecture that the heuristic could be in fact an optimal solution method for a problem with such a structure though a sound proof is yet to be investigated.

Now, Algorithm 2 is nested in Algorithm 1 for solving Model 2. The combined search algorithm has been coded in C + +.

5. Computational results and discussion

In this section, a 30-job problem is first solved to illustrate the application of the proposed adaptive approach. Then the ANG approach is compared with several other typical neighbourhood generation methods using 100 randomly generated problems of different sizes. Finally, the performance of the proposed method and effects of some search parameters are discussed. All the computations are run on the same 486 PC mentioned earlier.

5.1. Example

To illustrate, the proposed algorithm is applied to sequence 30 jobs on a machining center. Each job has its own due date. The deviation from the due date will result in earliness or tardiness penalty. The processing time of each job is controllable. The information about the processing times, due dates, compression and extension costs, as well as earliness and tardiness penalty costs is listed in Table 2. The sequence-dependent setup times are asymmetric and range between 0.5 to 8 min.

The solution to the example problem is listed in Table 3. The sequencing and processing time decisions can be made simultaneously based on the information in Table 3.

The computational results clearly indicate that the total cost associated with time compression, extension, and the penalty to the deviation from the due dates has been substantially reduced. For instance, if jobs in the example problem are processed within normal processing times and in the order shown in Table 2, the total processing cost would be \$5327. After the search, the total cost is reduced to \$504 (Table 3), representing a 90.5% improvement. Table 3 also shows that JIT sequencing has been achieved for 10 jobs, i.e., jobs 28, 16, 13, 3, 6, 26, 25, 20, 7, and 19. It is further shown that 24 out of the 30 jobs have a less than or equal to 10 min deviation from their due dates. However, if JIT sequencing is the predominant goal, the number of JIT-sequenced jobs can be increased by using the following model:

Model 3

$$\min_{1 \le s \le J!} G(s) = \operatorname{Min} \sum_{s=1}^{J!} W_s \sum_{\mu(s,l)=1}^{J} \left(a_{\mu(s,l)} T_{\mu(s,l)} + b_{\mu(s,l)} E_{\mu(s,l)} \right) \tag{27}$$

subject to constraints (2) to (9).

Model 3 can be solved using the same tabu search scheme as presented above with minor modification.

5.2. Discussion

To gain further insight into the problem and the proposed ANG approach, the discussion regarding the performance of the ANG method, cost reduction, and the effects of some search parameters is made with reference to additional computational results.

Table 2
Processing time, due date, and cost data for the example problem

j	P_j^N (min)	P_j^U (min)	P_j^L (min)	d_j (min)	a_j (\$/min)	b_j (\$/min)	α_j (\$/min)	β_j (\$/min)
1	3	4	3	72	5.0	5.0	4.0	1.0
2	4	5	3	360	4.0	3.0	5.5	1.6
3	8	14	8	300	4.0	3.0	4.5	1.2
4	8	20	8	30	1.5	1.5	1.8	1.4
5	10	10	10	270	6.5	1.5	4.0	1.0
6	10	15	8	330	4.0	4.0	2.6	0.3
7	10	18	10	432	6.0	3.0	1.5	2.5
8	10	15	8	390	4.5	2.0	2.0	2.0
9	10	12	5	180	5.0	3.0	2.7	1.5
10	11	15	9	210	4.5	1.5	1.7	1.4
11	12	12	9	228	1.5	1.0	3.5	2.5
12	12	12	10	420	2.0	0.5	0.9	0.3
13	14	20	10	210	4.0	3.0	0.5	0.5
14	14	14	5	465	3.5	0.5	3.6	2.3
15	15	15	15	115	1.0	1.0	1.0	1.3
16	15	20	12	150	8.5	0.5	1.5	0.5
17	15	19	7	210	2.5	1.0	0.6	0.3
18	15	25	11	282	3.0	0.5	3.0	0.2
19	15	25	8	480	8.0	1.0	1.0	0.5
20	18	20	14	405	4.0	1.5	0.6	0.3
21	20	25	15	30	5.5	2.0	2.3	1.3
22	20	28	14	120	6.0	2.0	1.0	0.6
23	20	30	12	240	5.5	2.0	3.5	1.5
24	20	30	11	480	4.5	1.0	1.8	1.0
25	24	25	12	360	7.0	4.0	1.8	0.4
26	25	40	14	345	5.0	2.0	2.2	0.5
27	28	35	15	60	5.0	1.0	2.0	0.5
28	30	50	20	90	5.5	1.5	1.6	0.2
29	30	37	18	240	3.5	2.5	1.5	2.0
30	35	40	22	144	5.0	1.5	0.8	0.5

Note: The total processing cost = \$5327 if jobs are sequenced in the order shown above.

5.2.1. Comparison of neighbourhood generation methods

In this section, the proposed ANG method is compared with four other commonly used methods. Here, only the results obtained using the best combinations of the search parameters (tabu-list size, diversification strategies, etc.) are compared. A total of 100 problems were randomly generated with 25 problems for each of the four problem sizes: $J = 30$, 40, 50, and 60 jobs. All problems were tested using the following neighbour generation and move selection methods:

Pairwise Interchange (PI) Method: This is the most basic approach. The transposition range in this case includes the entire permutation, i.e. from 1 to $J - 1$. A move is made to the best non-tabu neighbour in each iteration.

Adjacent Pairwise Interchange (API) Method [7]: With this method, only the adjacent jobs are switched. The size of a neighbourhood is $J - 1$. The move is selected in the same manner as in the PI method.

First Improving Neighbour (FIN) *Method* [5]: In this case, the neighbourhood is generated using the PI method. However, the first non-tabu move which improves the current value of the objective function is accepted and the neighbourhood generation process is truncated once such a move is found.

Restricted Neighbourhood (RN) Method: This method was proposed in [1]. For an iteration, r, only those

Table 3
Solution to the example problem obtained using the best diversification strategy and starting sequence

Sequence [a]	dP (min)	TE (min)	P^* (min)	C^* (min)
21	0	6	20	24
4	0	−4	8	34
27	0	−2	28	62
1	1	5	4	67
28	−8	0	22	90
22	2	6	22	114
30	−13	8	22	136
16	−3	0	12	150
15	0	−52	15	167
9	0	2	10	178
17	−8	22	7	188
10	−2	10	9	200
13	−4	0	10	210
29	−12	10	18	230
23	−8	−3	12	243
11	0	−28	12	256
5	0	1	10	269
18	0	−4	15	286
3	6	0	14	300
12	0	106	12	314
6	3	0	10	330
26	−11	0	14	345
25	−10	0	14	360
2	0	−6	4	366
8	5	7	15	383
20	2	0	20	405
14	0	45	14	420
7	0	0	10	432
24	8	17	28	463
19	0	0	15	480

Note: dP: Deviation from normal processing time; TE: Tardiness or earliness of a job; P^*: Final processing time; C^*: Final completion time.
[a] Total cost = $504 using this sequence.

Table 4
Comparison of different neighbourhood generation methods

Problem set [a]	Number of jobs J	Average cost reduction (%) [c]				
		PI	API	FIN	RN	ANG
1	30	90.6 [b]	77.8	64.5	91.4	93.1
2	40	69.5	59.0	42.3	84.7	92.0
3	50	51.4	41.2	29.6	62.3	83.2
4	60	25.3	16.3	19.2	30.2	62.9

[a] Each set contains 25 randomly generated problems.
[b] Average of 25 randomly generated problems.
[c] Cost reduction = (initial cost − final cost)/initial cost.

neighbours with a transposition range lower than ra(r) are evaluated and a move is then made to the best non-tabu neighbour in the neighbourhood. The function ra(r) is given by:

$$\mathrm{ra}(r) = \left[J(\mathrm{minra}/J)^{(r-\mathrm{ret})/\mathrm{estab}} \right]_{\mathrm{minra}}^{J-1}, \tag{28}$$

where minra is the minimum range of transpositions, ret the number of initial iterations in which the entire neighbourhood is evaluated, and estab the iteration number from which any transpositions greater than minra are not allowed. The following values are found to be best for our computations: minra = 4; rat = 1; estab = 15.

Adaptive Neighbourhood (ANG) *Method*: This is the proposed method in this study and the details have been elaborated in Section 4.3. The following ranges of the settings are used for the test: $\lambda_0 = 10\%$, $\psi = 0.01$ to 0.02, $\Psi = 0.004$ to 0.008, $R = 2$ to 4, RA(1) = $J - 1$, and ra(1) = $(1/2)J$ to $(2/3)J$.

For the comparison purpose, we have run the search using each of the methods for 10 min for the 30-job and 40-job problems and 15 min for the problems with 50 and 60 jobs. For each test problem, the tabu-list size is set to 10 moves and the same starting point was used for all the methods. When diversification was applied, the phase size was set at 20 moves. The average cost reductions are summarized in Table 4.

As shown in Table 4, the PI, RN, and ANG methods provide very good results for medium sized problems while the other two lag behind considerably. For example, for the 40-job problems, the PI, RN, and ANG methods can achieve cost reductions of 69.5%, 84.7%, and 93.1% respectively, but the cost reductions are only 59% and 42.3% respectively for the API and FIN approaches. As the problem size grows, the superiority of the ANG mechanism becomes more evident. For instance, after 15 min of search for the 60-job problems, it reduces the processing cost by an average of 63% while the second best, i.e., the RN method, leads to only about 30% cost reduction on the average. The low search efficiency of the PI, API, FIN, and RN methods is probably caused by either the neighbourhood generation or the move selection strategy. With the PI method, the neighbourhood size and hence the time required to make a move increases rapidly with the increased problem size. Though the neighbourhood size can be limited for the API, FIN, and RN methods, a large number of moves may be required to reposition jobs in the permutation.

The convergence curves of a typical 40-job problem corresponding to different neighbourhood generation and move selection mechanisms are plotted in Fig. 2. These curves show that the ANG method has the best improvement rate and it can achieve most of the cost reduction in the first 5 min. We have extended the computations to 24 h for the ANG method and no further improvement is observed.

Nevertheless, it should be mentioned that the RN and ANG approaches can be best applied to those problems which show a recognizable pattern in the disruption level (cost difference) with respect to the positions of jobs in the sequence. If such a tendency cannot be found, it may be advantageous to evaluate the entire neighbourhood or to perform other partial searches.

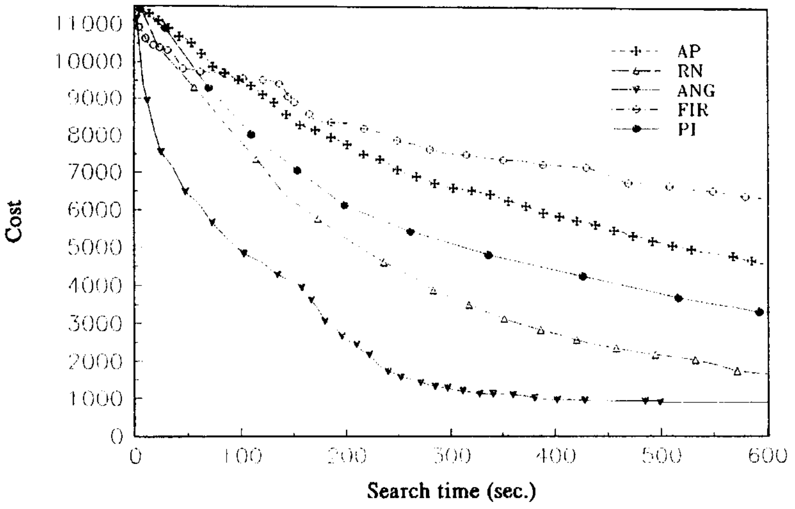

Fig. 2. Convergence curves for a 40 job problem using different neighbourhoud generation methods. (Note: Some data points have been omitted for presentation purposes.)

5.2.2. *Effects of search diversification and tabu-list size*

The average costs for the test problems obtained using the ANG method and different diversification strategies are summarized in Table 5. It is shown that search path diversification has some impact on the performance of the algorithm. The first diversification strategy, i.e., restarting from the best solution, *S_best*, marginally outperformed the third strategy, i.e. no diversification at all. A possible reason is that the search barriers are reduced due to the cleared tabu list after each search phase and thus the search can proceed more easily towards a better solution. The second diversification strategy, i.e., resuming the search from a randomly selected sequence after each phase, appears to be the worst of the three. The reason could be that part of the previous search effort is frequently discarded and thus the chance of resuming from a good solution is lower.

Computations have also been performed using tabu-lists of different sizes. It was observed that tabu lists of 5 moves are satisfactory for majority of the test problems but shorter tabu lists often lead to cycling. It was also found that the tabu lists containing 10 moves were sufficient to prevent cycling for all the tests problems. Our computational experience did not seem to favour longer tabu lists. Actually, tabu lists with more than 10 moves did not improve the results in our computational scope.

Table 5
Average costs for the final solutions with different diversification strategies (tabu-list size = 10, phase size = 20)

Problem set	Number ofjobs [a]	Diversification strategy		
		Restart form best sequence	Restart form random sequence	No diversification
1	30	617 [b]	759	624
2	40	874	2203	891
3	50	2045	3820	2052
4	60 [c]	7885	7885	7885

[a] Each set contains 25 randomly generated problems.
[b] Average of 25 randomly generated problems.
[c] Search did not complete first phase within the time limit.

6. Conclusion

In this paper, the problem of JIT sequencing of a machining center with variable processing times and sequence-dependent setups has been formulated as a non-linear mixed integer programming model. A tabu search algorithm based on an adaptive neighbourhood generation (ANG) method has been developed to efficiently solve the problem. Both job sequencing and processing time selection decisions can be made simultaneously based on the solution to the problem. Our computational results have shown that, as compared with four other neighbourhood generation methods, higher improvement rate can be achieved using the ANG method. This effect is more evident for larger problems.

The effects of search diversification strategies and tabu-list sizes are also investigated. It is found that restarting from the best found solution marginally improves the search performance. It is also observed that tabu lists longer than 10 moves have no significant effect on solution quality.

Acknowledgements

The authors wish to thank the two anonymous referees for their valuable suggestions and comments on this paper. Research funding from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

Appendix A

A.1. Proof of Theorem 1

(a) Referring to Eqs. (20)–(26), increasing processing time in the range of $[P^n_{\mu(s,k)}, P^n_{\mu(s,k)} + \Delta P^n_{\mu(s,k)}]$ will not change the values of $A_{\mu(s,k)}$, $B_{\mu(s,k)}$, and $G_{\mu(s,k)}$. Therefore, the saving will be a monotonic linear increasing function in the range with slope $F^n_{\mu(s,k)}$.

(b) Referring to step 3 of Algorithm 2, the magnitude of $F^n_{\mu(s,k)}$ is determined by one of the following cases:

Case A.1. If $\Delta P^n_{\mu(s,k)} = \delta_1$, i.e. $\Delta P_{\mu(s,k)} = E^n_{\mu(s,k')} = \min\{E^n_{\mu(s,l)} > 0 \mid l \geq k\}$ (see Eqs. (24) and (25)), any increment of processing time of job $\mu(s,k)$ greater than $E^n_{\mu(s,k')}$ will make the job in position k' tardy and therefore reduce $F^n_{\mu(s,k)}$ by at least $a_{\mu(s,k')} + b_{\mu(s,k')}$. This is because the original possible saving $b_{\mu(s,k')}$ has been lost and a new penalty cost, $a_{\mu(s,k')}$ is imposed due to this job's status change from early to tardy. In addition, the status change of the job in position k' may also cause status changes of other succeeding jobs in the sequence and thus additional reduction in $F_{\mu(s,k)}$ is possible.

Case A.2. If $\Delta P^n_{\mu(s,k)} = \delta_2$ (see Eqs. (24) and (26)), there exist the following two possibilities

(i) $V^n_{\mu(s,k)} > P^{\mathrm{U}}_{\mu(s,k)} - P^{\mathrm{N}}_{\mu(s,k)}$. If so, any increase of processing time of job $\mu(s,k)$ more than δ_2 will cause a transition from saving due to the released compression to penalty caused by extension and thus reduce $F_{\mu(s,k)}$ by at least $\alpha_{\mu(s,k)} + \beta_{\mu(s,k)}$. Since δ_2 is less than δ_1, increasing the processing time of job $\mu_{\mu(s,k)}$ by δ_2 will not affect the earliness or tardiness of other jobs. However, if further increasing the processing time of job $\mu(s,k)$, some succeeding jobs in the sequence may become tardy and thus additional reduction in $F_{\mu(s,k)}$ may occur as indicated in case 1.

(ii) $V^n_{\mu(s,k)} \leq P^{\mathrm{U}}_{\mu(s,k)} - P^{\mathrm{N}}_{\mu(s,k)}$. In this case, because $\Delta P^n_{\mu(s,k)} + P_{\mu(s,k)} = \delta_2 + P_{\mu(s,k)} = P^{\mathrm{U}}_{\mu(s,k)}$ (see Eq. (24)), i.e., the maximum allowed processing time has been reached, it is therefore impossible to increase processing time of job $\mu(s,k)$ by more than $\Delta P^n_{\mu(s,k)}$.

Theorem 1 is thus proved. □

A.2. Proof of Theorem 2

For a job $\mu(s,k)$, if its increasable processing time $V^n_{\mu(s,k)} > 0$ and its processing time is increased by $\Delta P^n_{\mu(s,k)} = \min\{\delta_1, \delta_2\}$, then for any other job $\mu(s,l)$, we have the following

Case A.3. $F^{n+1}_{\mu(s,l)} = F^n_{\mu(s,l)}$ if $\Delta P^n_{\mu(s,k)} = \delta_2$, or $\Delta P^n_{\mu(s,k)} = \delta_1$ $(\delta_1 = E^n_{\mu(s,k')} = \min\{E_{\mu(s,l)} > 0 \mid l \geq k\}$ and $l > k')$.

Case A.4. $F^{n+1}_{\mu(s,l)} = F^n_{\mu(s,l)} - (b_{\mu(s,k)} + a_{\mu(s,k)})$ if $\Delta P^n_{\mu(s,k)} = \delta_1$ $(\delta_1 = E^n_{\mu(s,k')} = \min\{E^n_{\mu(s,l)} > 0 \mid l \geq k\}$ and $l \leq k')$.

It follows that if the processing time of job $\mu(s,q)$ with maximum saving rate $F^n_{\mu(s,q)}$ in iteration n is increased by $\Delta P^n_{\mu(s,q)}$, and job $\mu(s,q')$ results in maximum saving rate $F^{n+1}_{\mu(s,q')}$ in iteration $n+1$, then $F^n_{\mu(s,q)} \geq F^{n+1}_{\mu(s,q')}$ always holds. This leads to

$$F^n_{\mu(s,q)} \geq F^m_{\mu(s,q')} \quad (m > n). \tag{A.1}$$

Now, suppose the solution obtained using Algorithm 2 can be further improved in at least one future iteration, say, iteration m $(m > n)$. Then there must exist an $F^m_{\mu(s,q')} = \max\{F^n_{\mu(s,l)} \mid l = 1, \ldots, J\} > 0$, which means $F^m_{\mu(s,q')} > F^n_{\mu(s,q)}$. This, however, contradicts (A.1) and thus completes the proof of Theorem 2. □

References

[1] B. Adenso-Diaz, Restricted neighbourhood in the tabu search for the flowshop problem, European Journal of Operational Research 62 (1992) 27–37.

[2] K.R. Baker, G.D. Scudder, Sequencing with earliness and tardiness penalties: A review, Operations Research 38 (1) (1990) 22–36.

[3] T.C.E. Cheng, Z.L. Chen, L. Chung, Parallel-machine scheduling with controllable processing times, IIE Transactions 28 (1996) 177–180.

[4] T.C.E. Cheng, M.C. Gupta, Survey of scheduling research involving due date determination decisions, European Journal of Operational Research 38 (1989) 156–166.

[5] H.A.J. Crauwels, C.N. Potts, L.N. Wassenhove, Local search heuristics for single-machine scheduling with batching to minimize the number of late jobs, European Journal of Operational Research 90 (1996) 200–213.

[6] R.L. Daniel, R.K. Sarin, Single machine scheduling with controllable processing times and number of jobs tardy, Operations Research 37 (6) (1989) 981–984.

[7] F. Della Croce, Generalized pairwise interchanges and machine scheduling, European Journal of Operational Research 83 (1995) 310–319.

[8] E.A. Elsayed, M.-K. Lee, S. Kim, E. Scherer, Sequencing and batching procedures for minimizing earliness and tardiness penalty of order retrievals, International Journal of Production Research 31 (3) (1993) 727–738.

[9] J.R., Evans, D.R. Anderson, D.J. Sweeney, T.A., Williams, Applied Production and Operations Management, West Publishing Company, New York, 1990.

[10] M.R. Garey, R.E. Tarjan, G.T. Wilfong, One-processor scheduling with symmetric earliness and tardiness penalties, Mathematics of Operations Research 13 (2) (1988) 330–348.

[11] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13 (1986) 533–549.

[12] F. Glover, Tabu search: A tutorial, Interfaces 20 (4) (1990) 74–94.

[13] S.K. Gupta, J. Kyparisis, Single machine scheduling research, Omega 15 (3) (1987) 207–227.

[14] R. Hubscher, F. Glover, Applying tabu search with influential diversification to multiprocessor scheduling, Computers and Operations Research 21 (8) (1994) 877–884.

[15] J. Knox, Tabu search performance on the symmetric travelling salesman problem, Computers and Operations Research 21 (8) (1994) 867–874.

[16] I. Lee, Single machine scheduling with controllable processing times: A parametric study, International Journal of Production Economics 22 (2) (1991) 105–110.

[17] C.Y. Lee, S.J. Kim, Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights, Computers and Industrial Engineering 28 (2) (1995) 231–243.

[18] J.K. Lenstra, A.H.G. Rinnoy, P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343–362.

[19] C. Oguz, C. Dincer, Single machine earliness-tardiness scheduling problems using the equal-slack rule, Journal of Operational Research Society 45 (5) (1994) 589–594.

[20] S.C. Sarin, E. Erel, G. Steiner, Sequencing jobs on a single machine with common due date and stochastic processing times, European Journal of Operational Research 51 (2) (1991) 188–198.

[21] R.G. Vickson, Two single machine sequencing problems involving controllable job processing times, AIIE Transactions 12 (3) (1980) 258–262.

[22] H.K.B. William, Experimental investigation of intelligent search methods for job scheduling, Ph.D. Dissertation, Faculty of Operations Management, Georgia Institute of Technology, 1994.

[23] D.L. Woodruff, Simulated annealing and tabu search: Lessons from a line search, Computers and Operations Research 21 (8) (1994) 829–831.