INT. J. PROD. RES., 01 JUNE 2004, VOL. 42, NO. 11, 2289–2301

Scheduling in flowshops to minimize total tardiness of jobs

SAMEER HASIJA and CHANDRASEKHARAN RAJENDRAN*

The problem of scheduling in static flowshops is considered with the objective of minimizing mean or total tardiness of jobs. A heuristic algorithm based on the simulated annealing (SA) technique is developed. The salient features of the proposed SA algorithm are the development of two new perturbation schemes for use in the proposed SA algorithm and a new improvement scheme to improve the quality of the solutions. The proposed algorithm is evaluated by using the benchmark problems available in the literature. The performance of the proposed SA algorithm is found to be very good, and the proposed heuristic performs better than the existing heuristics.

1. Introduction

A flowshop is a conventional manufacturing system where all machines are arranged in the order of operations performed on jobs, and the operation sequence is the same for all jobs. The flowshop scheduling problem has been a keen area of research for many years. As a vast majority of flowshop scheduling problems are either NP-complete or NP-hard, research is mostly directed towards the development of heuristic or near-optimal methods. The objective of minimizing makespan has been the objective of many heuristic methods developed so far. Some of the noteworthy heuristics for minimizing makespan have been developed by Campbell *et al.* (1970), Dannenbring (1977), Nawaz *et al.* (1983), Widmer and Hertz (1989), Leisten (1990), Ogbu and Smith (1990), Ishibuchi *et al.* (1995), Rajendran (1995), Nowicki and Smutnicki (1996), Rajendran and Ziegler (1997), Ben-Daya and Al-Fawzan (1998), and Framinan *et al.* (2001). Research has also concentrated on the development of algorithms with the objective of minimizing total or mean tardiness of jobs owing to its significance and importance to real-life considerations. There have been four attempts to develop heuristics for solving the flowshop scheduling problem with the objective of minimizing total tardiness of jobs. The first attempt was by Gelders and Sambandam (1978). They developed constructive heuristics. Later, Kim (1993) proposed a heuristic that was based on the tabu search procedure of Widmer and Hertz (1989). Parthasarathy and Rajendran (1998) proposed two heuristics based on the simulated annealing technique. They compared their heuristics with those by Gelders and Sambandam, and Kim, by generating a large number of problems of various sizes. The heuristics by Parthasarathy and Rajendran were found to perform better than those by Gelders and Samdandam, and Kim. In a parallel and independent work, Armentano and Ronconi (1999) proposed a heuristic, based on the tabu search procedure. It is evident from the

Revision received June 2003

Industrial Engineering and Management Division, Department of Humanities and Social Sciences, Indian Institute of Technology Madras, Chennai – 600036, India.

*To whom correspondence should be addressed. e-mail: craj@iitm.ac.in

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540310001657595

literature review that the heuristics by Parthasarathy and Rajendran (1998), and Armentano and Ronconi (1999) are the best-performing heuristics to minimize the total tardiness of jobs. Hence, in the present study we have chosen to include these heuristics in the performance evaluation.

In this paper, we consider the development of a heuristic algorithm with the objective of minimizing the total tardiness of jobs in flowshops. The best-performing existing heuristics by Parthasarathy and Rajendran (1998) and Armentano and Ronconi (1999), and the proposed heuristic, are compared by using the benchmark problems of Taillard (1993).

2. Problem formulation

2.1. *Notation*

t_{ij}	processing time of job i on machine j.
D_i	due date of job i.
n	total number of jobs to be scheduled.
m	total number of machines in the flowshop.
σ	ordered set of jobs already scheduled, out of n jobs; a partial sequence.
$q(\sigma, j)$	completion time of partial sequence σ on machine j (i.e. release time of machine j after processing all jobs in partial sequence σ).
$q(\sigma i, j)$	completion time of job i on machine j, when the job is appended to partial sequence σ.

In studying flowshops it is usually assumed that the job sequence is to be chosen to be identical on each of the machines. A schedule of this type is called a permutation schedule, and in this paper, permutation schedules are considered. We are also considering static flowshop scheduling problems, i.e. all jobs are assumed to be available at the beginning of the scheduling period.

For calculating the completion time of jobs on machines in a flowshop, recursive equations are used as follows.

Initialize $q(\sigma i, 0)$, the completion time of job i on machine 0, equal to zero. This time indicates the time availability of a job in the flowshop, and it is equal to 0 for all jobs in the case of static flowshops. Also, $q(\phi, j)$ is set to zero for all js, where ϕ denotes a null sequence.

For $j = 1$ to m do

$$q(\sigma i, j) = \max\{q(\sigma, j);\ q(\sigma i, j-1)\} + t_{ij}. \tag{1}$$

$$\text{Set} \quad C_i = q(\sigma i, m), \tag{2}$$

where C_i denotes the completion time of job i on machine m. When all jobs are scheduled, we have σ denoting the complete set of n jobs, and hence we have

$$\text{makespan } (M) = \max\{q(\sigma i,\ m),\ i = 1, 2, 3, \ldots, n\}, \text{ and} \tag{3}$$

$$\text{total tardiness } (TARD) = \sum_{i=1}^{n} \max\{(q(\sigma i,\ m) - D_i);\ 0\}. \tag{4}$$

3. Mechanics of the proposed heuristic

The proposed heuristic is based on the simulated annealing (SA) technique. A detailed discussion of the heuristic is presented in this section.

3.1. *Setting and updating of parameters used in the SA algorithm*

The initial temperature T is set to 540. A counter called *FREEZE* is used to terminate the algorithm. Let S denote the seed sequence, let S'' denote the best sequence obtained after performing the two proposed perturbation schemes on S, and let B refer to the best sequence so far generated by the proposed SA algorithm with respect to the objective function. *COUNT* and *REPLACE* denote the counters that keep track of the total number of sequences generated and the total number of accepted moves, respectively, at a particular temperature. These counters are set to zero at the start of every temperature.

Assume that S'' is the best sequence among all sequences obtained by employing the perturbation schemes on S at some particular temperature T. Let the objective function value of S, S'' and B be Z, Z'' and Z_B, respectively.

```
If Z'' ≤ Z
    then
        {
        set S ⇐ S'';
        REPLACE is incremented by 1;
        If Z'' ≤ Z_B   /* S'' is now compared with the best sequence obtained
                          so far */
            then
                {
                set B ⇐ S'';
                reset FREEZE to zero;
                COUNT is incremented by 1;
                }
            else
                COUNT is increased by 1;
        }
    else
        {
        assign P = exp(−((Z'' − Z) × 100 /Z)/T);
        sample a uniform random number u in the range (0, 1);
        if u ≤ P
                then
                    {
                        set S ⇐ S'';
                        increment COUNT by 1;
                        increment REPLACE by 1;
                    }
                else
                        S'' is rejected, S is retained, and increment COUNT
                        by 1.
        }
```

The iterations at a particular temperature are terminated when *COUNT* is greater than or equal to $(2 \times n)$. Hence the epoch length is controlled by the number of generated solutions (equal to $2 \times n$) at a given temperature. After the iterations are over at a particular temperature, the percentage of accepted moves is calculated as $(REPLACE \times 100)/COUNT$. If it is less than 15%, then *FREEZE* is

increased by 1. The temperature for the next iteration is obtained by multiplying the present temperature by a factor of 0.9. The termination of the SA algorithm is done when either *FREEZE* is equal to 5 or the temperature is less than or equal to 20. This is done because the chances of obtaining further good solutions (that are better than the best solution obtained so far) appear to be diminishing, and hence we terminate the search process.

It should be noted that if the initial value of temperature is chosen to be high, then very inferior solutions will be accepted. We proceed with the assumption of accepting solutions inferior by 50% with the probability of 0.9 in the first iteration. Hence we arrive at the initial temperature of about 475 (i.e. by setting $0.9 = \exp(-50/T)$, we get $T \approx 475$). In order to obtain an efficient initial temperature setting, a few trial runs were made with starting temperatures in the neighborhood of 475. The test results were best with the initial temperature setting of 540 during the trial runs. The final temperature is fixed at 20, as it makes sure that the algorithm runs for 32 temperature settings, which is found to be a good balance between computational effort and solution quality in the pilot runs.

We now present more details of the proposed heuristic.

3.2. *Seed solution*

To present the methodology used to generate the initial seed sequence, we present the following example.

Let us consider a hypothetical small-sized flowshop problem:

	Machine 1	Machine 2	Machine 3	Due-date D_i
Job 1	2	3	2	14
Job 2	5	1	4	16
Job 3	5	3	2	10

Here D_i is the due date of job i, and t_{ij} denotes the processing time of job i on machine j.

We compute

$$D'_{i1} = (D_i \times t_{i1}) \Big/ \sum_{j=1}^{m} t_{ij} \tag{5}$$

$$D'_{ij} = D'_{i,j-1} + (D_i \times t_{ij}) \Big/ \sum_{k=1}^{m} t_{ik}\ , \text{ for } j = 2, 3, \ldots, m. \tag{6}$$

For the above example D'_{ij} values are given below:

	Machine 1	Machine 2	Machine 3
Job 1	4	10	14
Job 2	8	9.6	16
Job 3	5	8	10

Now we arrange the jobs in ascending order of D'_{ij} values for every machine j and hence we get m sequences corresponding to m machines. In this case we get sequences

{1-3-2}, {3-2-1} and {3-1-2}. We compute the total tardiness of these sequences, which are 3, 3 and 1 respectively. We can now choose sequence {3-1-2} as the seed sequence for the SA algorithm because this sequence has the least value of total tardiness.

The solution yielded by this method is subjected to an improvement scheme called the job-index-based insertion scheme (JIBIS). This scheme is described as follows.

3.3. *Job-index-based-insertion-scheme (JIBIS) for improvement of a sequence*

In a five-job problem suppose we have sequence {2-1-4-5-3} to be improved upon. Set this sequence as the seed sequence. Take job 1 and insert in all possible positions to get {1-2-4-5-3}, {2-4-1-5-3}, {2-4-5-1-3} and {2-4-5-3-1}. Choose the best sequence; say, {2-4-1-5-3} is chosen. If it is better than the seed sequence {2-1-4-5-3}, then update the seed sequence as {2-4-1-5-3}; else retain the seed sequence as {2-1-4-5-3}. Assume that the current seed sequence is updated as {2-4-1-5-3}. Take job 2 and insert it in various positions to obtain {4-2-1-5-3}, {4-1-2-5-3}, {4-1-5-2-3} and {4-1-5-3-2}. Choose the best sequence; say, {4-2-1-5-3} is chosen. If it is better than {2-4-1-5-3} then update the seed sequence as {4-2-1-5-3}. Suppose sequence {2-4-1-5-3} is better than sequence {4-2-1-5-3}. Hence the sequence {2-4-1-5-3} is retained as the seed sequence. Likewise all other jobs (i.e. jobs 3, 4 and 5) are chosen for insertion and the seed sequence is updated. Note that the final solution yielded by JIBIS is always better than or equal to the quality of the solution yielded by the seed sequence.

3.4. *Perturbation schemes*

Two perturbation schemes are proposed in this paper for use in the SA technique.

3.4.1. *Job-shift-based (JSB) perturbation scheme*

This scheme chooses a job in the seed sequence based on a probabilistic function and then inserts the chosen job either to the right or to the left of its original position in the sequence. If a job in position i is chosen, then a random number between 0 and 1 is subsequently generated. If this number is greater than 0.5, then the job is inserted in any position to its right in the sequence; else the job is inserted to any position to its left in the original sequence. The basis on which the job to be inserted is chosen is the relative displacement of the job's position in the present sequence from the job's position in the best sequence obtained so far. For example, assume that we have the seed sequence {2-3-1-5-4} to be perturbed and the best sequence obtained so far in the SA heuristic as {1-4-2-5-3}. The shift of the position of job i is defined as follows:

$$shift(i) = \left| P_p(i) - P_b(i) \right| + 1, \tag{7}$$

where $P_p(i)$ gives the position of job i in the present sequence, and $P_b(i)$ gives the position of job i in the best sequence obtained so far. For example, we have $P_p(1) = 3$ and $P_b(1) = 1$ and hence $shift(1) = 3$. Thus we have $shift(2) = 3$, $shift(3) = 4$, $shift(4) = 4$, and $shift(5) = 1$.

The relative shift index of job i (i.e. $RS(i)$) is defined as

$$RS(i) = shift(i) \Big/ \sum_{i=1}^{n} shift(i). \tag{8}$$

Hence, we have $RS(1) = 3/15$, $RS(2) = 3/15$, $RS(3) = 4/15$, $RS(4) = 4/15$ and $RS(5) = 1/15$.

The relative shift can be regarded as the probability of a job to be picked up for insertion on the basis of the position of job i in the sequence to be perturbed, relative to its position in the best sequence. Now sample a uniform random number, and assume that the random number generated is 0.6; hence job 3 is chosen for insertion. Next, we generate a uniform random number. Assume we get 0.8; thus a position to the right of job 3 has to be randomly chosen for insertion. Suppose the position 4 is chosen; we obtain {2-1-5-3-4} as the sequence generated by the JSB perturbation scheme.

The rationale of the perturbation scheme is as follows. The scheme selects with higher probability the job which has the maximum shift in its position (relative to its position in the best sequence) and inserts the job in another position probabilistically. Basically, the job with minimum 'displacement' in its position (relative to its position in the best sequence) is less likely to be considered for insertion in other positions, and the job with maximum 'displacement' in its position (relative to its position in the best sequence) is more likely to be considered for insertion in other positions. This means that a job can be chosen more than once for insertion, as against the CRIPS, a perturbation scheme used by Parthasarathy and Rajendran (1998). It also means that the best sequence guides the choice of the job taken up for insertion, and in this way we attempt to improve upon the best sequence that has been obtained so far in the SA algorithm.

3.4.2. *Probabilistic-step-swap (PSS) perturbation scheme*

This scheme works as follows, starting with a seed sequence:

Step 1: Assign $a = 1$ and $b = 2$.
Step 2: Generate a uniform random number between 0 and 1.
Step 3: If the number generated is greater than 0.67

then
{
generate a new sequence by swapping the jobs in positions a and b of the seed sequence; set the newly generated sequence as the seed sequence; and increment b by one.
}
else
increment both a and b by one.

Step 4: If $b < n + 1$

then
go back to step 2;
else
proceed to step 5.

Step 5: Stop, if at least one sequence that is different from the original seed sequence has been generated in the process; else go back to Step 1.

The perturbation scheme generates a new sequence from a seed sequence by probabilistically swapping jobs that are close to each other; however, once such a sequence is generated, a subsequent sequence is generated such that farther jobs are chosen for swapping probabilistically. Hence we find that sequences that are nearer as well as farther from the original seed sequence (in respect of variable-space of the scheduling problem) are generated by this perturbation scheme. For example, consider the seed sequence to be perturbed as {2-4-3-1-5}. Assume the uniform random number generated to be 0.75; we now get a new sequence {4-2-3-1-5} by swapping jobs [1] and [2]. We now have $a=1$ and $b=3$, with the seed sequence being {4-2-3-1-5}. Sample a uniform random number. Assume the next uniform random number generated is 0.97; we get the next new seed sequence as {3-2-4-1-5}. We have $a=1$ and $b=4$. Sample a uniform random number. Assume the uniform random number is 0.21; then there is no perturbation in this step. Set $a=2$ and $b=5$. Sample a uniform random number. If the next uniform random number is 0.81, then we get {3-5-4-1-2} as the sequence after perturbation. The best sequence (with respect to minimum total tardiness of jobs) is chosen from all these generated sequences.

It should be noted that if two jobs in position a and b are swapped, then b is incremented by 1, and jobs now present in positions a and b are considered for swapping. If jobs in positions a and b are not swapped, then both a and b are incremented by 1. It is evident that the same pair of jobs is not considered twice for swapping. Hence, the number of generated sequences is in the range $[1, n-1]$, by using this perturbation scheme.

3.5. *Obtaining the final (best) solution*

An archive of the ten best sequences that have been obtained in the SA search process is maintained. These ten best sequences, at the completion of the annealing process, are each subjected to the job-index-based insertion scheme (JIBIS). The best sequence among the sequences thus generated is the sequence yielded by the proposed SA algorithm as the final heuristic solution. It should be noted that the ten best sequences are subjected to the JIBIS in order to attempt to reach the point of global minimum around these points of local minima, as the solutions in the archive may or may not correspond to the global minimum.

4. The proposed SA algorithm

We present the step-by-step procedure of the SA algorithm proposed in this paper.

4.1. *Notation and terminology*

FREEZE	counter used to terminate the algorithm.
REPLACE	counter to keep track of the total number of accepted moves at the temperature T.
COUNT	counter to keep track of total number of iterations generated at the temperature T.
PER	the percentage of accepted moves at a particular temperature.
B	the best sequence obtained so far in the search process.
S	sequence on which JSB and PSS perturbation schemes are implemented.
S'	a sequence obtained by perturbing S.

S''	best sequence obtained amongst all sequences generated by JSB and PSS schemes.
Z, Z', Z'', Z_B	respective objective function values of S , S', S'' and B.
$JSB(S, S', Z')$	routine using JSB perturbation scheme with S as input, and S' as output with Z' as the objective function value of S'.
$PSS(S, S', Z')$	routine using PSS perturbation scheme with S as input, and S' as output with Z' as the objective function value of S'.
$DEL(B, S'')$	the routine for computing relative improvement (or deterioration) of S'' with respect to B. We have the relative measure given by $$DELTA\_B = (Z'' - Z_B) \times 100/Z_B. \quad (9)$$
$DEL(S, S'')$	the routine for computing the relative improvement (or deterioration) of S'' with respect to S. We have the relative measure given by $$DELTA = (Z'' - Z) \times 100/Z. \quad (10)$$
P	acceptance probability given by $\exp(-DELTA/T)$.
u	a uniform random number.

4.2. *Step-by-step procedure of the SA algorithm*

Step 1: Obtain the initial seed sequence using the method presented in Section 3.2. Assign this sequence to both S and B.

Step 2: Perform the improvement scheme JIBIS (presented in Section 3.3) on S and B.

Step 3: Compute the objective function value for the resultant sequences S and B, and assign these values to Z and Z_B, respectively.

Step 4: Initialize $T = 540$, $COUNT = 0$, $REPLACE = 0$ and $FREEZE = 0$.

Step 5: Generate and evaluate nine random sequences with respect to total tardiness of jobs, and store them and B in an archive of best ten sequences.

Step 6: If ($FREEZE = 5$) or ($T \leq 20$)
then
go to Step 21;
else
proceed to Step 7.

Step 7: Set $Z'' =$ a large number.

Step 8: Do the following steps twice:

{
Do the following steps n times:
{
Invoke perturbation scheme $JSB(S, S', Z')$;

If Z' is less than the maximum value of the objective function of a sequence amongst all sequences in the archive, then replace that sequence and its objective-function value by S' and Z';

Also, if Z' is less than Z'', then set $S'' \Leftarrow S'$ and $Z'' \Leftarrow Z'$.
}
}

Step 9: Do the following steps twice:

{

Invoke perturbation scheme $PSS(S, S', Z')$;

If Z' is less than the maximum value of the objective function of a sequence amongst all sequences in the archive, then replace that sequence and its objective-function value by S' and Z' respectively;

Also, if Z' is less than Z'', then set $S'' \Leftarrow S'$ and $Z'' \Leftarrow Z'$.

}

Step 10: Compute $DELTA$ from $DEL(S, S'')$.

Step 11: If $DELTA \leq 0$
then
proceed to Step 12;
else
go to Step 15.

Step 12: Set $S \Leftarrow S''$, $Z \Leftarrow Z''$ and $REPLACE = REPLACE + 1$.

Step 13: Invoke routine $DEL(B, S'')$ and compute $DELTA\_B$.
If $DELTA\_B \leq 0$
then
proceed to Step 14;
else
go to Step 17.

Step 14: Assign $B \Leftarrow S''$, $Z_B \Leftarrow Z''$ and $FREEZE = 0$, and go to Step 17.

Step 15: Compute P and sample u, a uniform random number in the range $(0, 1)$.
If $u > P$
then
go to Step 17;
else
proceed to Step 16.

Step 16: Set $S \Leftarrow S''$, $Z \Leftarrow Z''$ and $REPLACE = REPLACE + 1$.

Step 17: Set $COUNT = COUNT + 1$.

Step 18: If ($COUNT > 2n$)
then
proceed to Step 19;
else
go back to Step 7.

Step 19: Compute $PER = (REPLACE \times 100/COUNT)$;
If $PER \leq 15$
then
set $FREEZE = FREEZE + 1$ and proceed to Step 20;
else
proceed to Step 20.

Step 20: Set $T = T \times 0.9$, $COUNT = 0$ and $REPLACE = 0$, and go back to Step 6.

Step 21: Implement the improvement scheme JIBIS on all sequences stored in the archive. The best sequence, amongst all the sequences generated by the implementation of JIBIS, is returned as the heuristic solution. STOP.

5. Performance analysis

A comparison of the proposed heuristic (denoted by SRH) is made with the best-existing heuristics, namely, the tabu search heuristic by Armentano and Ronconi (1999) (denoted by ARH in the present study) and the simulated annealing heuristic by Parthasarthy and Rajendran (1998) (denoted by PRH in the current study). The heuristic by Parthasarthy and Rajendran (1998) is based on the SA technique. The salient features of this algorithm are the two perturbation schemes, the random insertion perturbation scheme (RIPS) and the curtailed random insertion perturbation scheme (CRIPS). RIPS can be explained by the following illustration. Consider a seed sequence {1-3-5-2-4}. Call this sequence S. The job in the first position can be inserted at any position to its right. Hence the job in the first position is inserted at any position between 2 and n (here $n=5$) randomly, and a random number generated between 2 and n is used to select the job position. Suppose the selected position is 4. Job [1] is inserted in position 4, yielding a new sequence S'_1 as {3-5-2-1-4}. Consider the job in the second (i.e. non-extreme) position of the seed sequence and choose randomly two positions for its insertion. Note that this job can be inserted at any position between the third and the nth positions (i.e. a position to its right) and in position 1 (i.e. a position to its left). Suppose position 1 is chosen to its left and position 3 to its right. The new sequences thus generated are {3-1-5-2-4} and {1-5-3-2-4}. Call these S'_2 and S'_3. The same procedure can be continued for all the jobs in positions 2 through $(n-1)$ in the original sequence of {1-3-5-2-4}. For the job in position n, one position can be selected towards its left, i.e. between positions 1 and $(n-1)$. Thus for a particular seed sequence with n jobs, $2\times(n-1)$ different sequences are generated. CRIPS is similar to RIPS except that the jobs in positions between 2 and $(n-1)$ are shifted either to the left or to the right with equal probability. For this purpose, a random number is generated between 0 and 1. If the random number is greater than 0.5, then a position to the right of the job's original position is chosen; else a position to the left of the job's original position is chosen. For the purpose of relative evaluation of the proposed SA algorithm, the SA heuristic by Parthasarthy and Rajendran (1998) that makes use of RIPS is considered in this study.

The heuristic proposed by Armentano and Ronconi (1999) is based on tabu search with the objective of minimizing total tardiness of jobs in flowshop scheduling problems. In this heuristic, every job, other than those in the tabu list, is inserted in every position, except in its original position in the given sequence, to generate the neighbourhood solutions. For every given solution, all such neighbouring solutions are evaluated and the best solution is selected. A tabu restriction with one job is maintained. The restriction is dependent on a tabu tenure, which is dynamically updated.

It is to be noted that the proposed algorithm enumerates the number of sequences as $(267n^2-139n)$ in the worst case and about $(31n^2+9n)$ in the best case. The PRH and the ARH are made to run so that the number of solutions searched in the solution domain are equal to or greater than the number of solutions searched by SRH. This is done in order to evaluate all heuristics with reference to almost the same computational effort.

The test problems chosen for evaluating the heuristics under study are the benchmark problems proposed by Taillard (1993). Though these problems have been proposed for evaluating heuristics with the objective of minimizing makespan, it is quite common to use these problems for evaluating heuristics with other objectives as well (e.g. see Liu and Reeves (2001) with the consideration of minimizing total flowtime of jobs). For computing the due dates the following procedure is used.

Step 1: Compute the sum of the processing times of job i, i.e.

$$T_i' = \sum_{j=1}^{m} t_{ij}. \tag{11}$$

Step 2: The due date of job i is given by

$$D_i = T_i' \times [1 + u \times 3], \tag{12}$$

where u is a uniform random number, generated using the routine given by Taillard (1993), in the range [0, 1]. It is evident that the due dates of jobs, thus determined, are tight enough to make many jobs tardy, and hence a good performance evaluation of heuristics in terms of minimizing the total tardiness of jobs can be done.

The relative percentage increase, RPI_i for heuristic i, is given by

$$RPI_i = (TARD_i - TARD_{\min}) \times 100/TARD_{\min}, \tag{13}$$

where $TARD_i$ is the tardiness of the heuristic for which the RPI is being evaluated, and $TARD_{\min}$ is the minimum tardiness value yielded by the three heuristics under study.

Table 1 gives the results of the relative performance of the proposed SA algorithm and the best existing heuristics. It is evident that the proposed SA algorithm performs the best amongst the heuristics under evaluation.

Table 1. Relative performance of the heuristics with the objective of minimizing total tardiness of jobs.

Number of jobs n	Number of machines m	Mean relative percentage increase in tardiness		
		SRH	PRH	ARH
20	5	0.2379	3.2520	5.3921
	10	0.3040	7.9124	5.4899
	20	0	2.8122	1.3681
50	5	0.4074	1.1978	3.4615
	10	0	2.6447	4.1825
	20	0.3805	5.7629	7.4865
100	5	0.0966	0.7744	2.0679
	10	0.0957	1.4236	3.6138
	20	0.1065	2.1955	2.1208

Note:
1. The number of problems given in Taillard (1993) for the given problem size ($n \times m$) is 10.
2. SRH refers to the heuristic proposed in this study, PRH refers to the heuristic proposed by Parthasarthy and Rajendran (1998), and ARH refers to the heuristic by Armentano and Ronconi (1999).

6. Summary

This paper has addressed the problem of scheduling in flowshops with the objective of minimizing total tardiness. The salient feature of the proposed SA algorithm is the development of two new perturbation schemes that have been incorporated in the basic SA algorithm. In addition, an improvement scheme to improve the seed solution that is given as the input to the SA is presented, an archive containing the best ten sequences obtained from the SA is maintained, and all such sequences are subjected to the improvement scheme at the end of the SA algorithm to improve the final solution. A relative evaluation of the proposed SA algorithm with the best existing SA and tabu search heuristics showed the superior performance of the proposed SA algorithm.

Acknowledgement

The authors are grateful to the referee for suggestions and comments that improved the earlier version of the paper.

References

Armentano, V. A. and Ronconi, D. P., 1999, Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & Operations Research*, **26**, 219–235.

Ben-Daya, M. and Al-Fawzanm, 1998, A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, **109**, 88–95.

Campbell, H. G., Dudek, R. A. and Smith, M. L., 1970, A heuristic algorithm for the *n*-job, *m*-machine problem. *Management Science*, **16**, B630–B637.

Dannenbring, D. G., 1977, An evaluation of flow-shop sequencing heuristics. *Management Science*, **23**, 1174–1182.

Framinan, J. M., Ruiz-Usano, R. and Leisten, R., 2001, Sequencing CONWIP flow-shops: analysis and heuristic. *International Journal of Production Research*, **39**, 2735–2749.

Gelders, L. F. and Sambandam, N., 1978, Four simple heuristics for scheduling a flowshop. *International Journal of Production Research*, **16**, 221–231.

Ignall, E. and Schrage, L., 1965, Application of the branch-and-bound technique to some flowshop scheduling problems. *Operations Research*, **13**, 400–412.

Ishibuchi, H., Misaki, S. and Tanaka, H., 1995, Modified simulated annealing algorithms for the flow shop sequencing problems. *European Journal of Operational Research*, **81**, 388–398.

Johnson, S. M., 1954, Optimal two- and three-stage production schedules. *Naval Research Logistics Quarterly*, **1**, 61–68.

Kim, Y. D., 1993, Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of Operational Research Society*, **44**, 19–29.

Leisten, R., 1990, Flow-shop sequencing problems with limited buffer storage. *International Journal of Production Research*, **28**, 2085–2100.

Liu, J. and Reeves, C. R., 2001, Constructive and composite heuristic solutions to the $P//\Sigma C_i$ scheduling problem. *European Journal of Operational Research*, **132**, 439–452.

Nawaz, M., Enscore Jr., E. E. and Ham, I., 1983, A heuristic algorithm for the *m*-machine, *n*-job flowshop sequencing problem. *OMEGA*, **11**, 95–95.

Nowicki, E. and Smutnicki, C., 1996, A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, **91**, 160–175.

Ogbu, F. A. and Smith, D. K., 1990, The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem. *Computers and Operations Research*, **17**, 243–253.

Parthasarthy, S. and Rajendran, C., 1998, Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. *Computers & Industrial Engineering*, **34**, 531–546.

Rajendran, C., 1995, Heuristic for scheduling in flowshop with multiple objectives. *European Journal of Operational Research*, **82**, 540–555.

RAJENDRAN, C. and ZIEGLER, H., 1997, Heuristics for scheduling in flowshop with setup, processing and removal times separated. *Production Planning & Control*, **8**, 568–576.

TAILLARD, E., 1993, Benchmark for basic scheduling problems. *European Journal of Operational Research*, **64**, 278–285.

WIDMER, M. and HERTZ, A., 1989, A new heuristic method for flowshop sequencing problem. *European Journal of Operational Research*, **41**, 186–193.