Applied Mathematics and Computation 217 (2010) 557–567

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Tabu search for non-permutation flowshop scheduling problem with minimizing total tardiness

Li-Man Liao *, Ching-Jen Huang

Department of Industrial Engineering and Management, National Chin-Yi University of Technology, Taichung County 41101, Taiwan

ARTICLE INFO

Keywords:
Scheduling
Non-permutation flowshop
Tardiness
MILP model
Tabu search

ABSTRACT

This paper deals with the non-permutation flowshop problem which means that the job sequences are allowed to be different on machines. The objective function is minimizing the total tardiness. Firstly, three mixed-integer linear programming (MILP) models for non-permutation flowshop problems are described, and then are analyzed and assessed their relative effectiveness. Secondly, two Tabu search based algorithms, denoted by LH1 and LH2, are proposed to solve the complicated non-permutation flowshop problems. The algorithms are constructed on specific neighborhood structures which enable the searching method effective. Finally, the performance is evaluated against Taillard's famous benchmark. Computational experiments show that the proposed algorithms, LH1 and LH2, are significantly superior to the L_TS algorithm. And the percentages of improved permutation flowshop instances by LH1 and LH2 algorithms are about 87.8% and 98.3% with respect to total tardiness, respectively. The non-permutation schedules also have achieved significant improvement in four different scenarios of due dates. Consequently, average percentage improvement (API) is 14.52% for flowshop problems with low tardiness factors. It is evident that exploring non-permutation schedule is effective and necessary for low tardiness factors.

1. Introduction

A flowshop is a conventional manufacturing system where all jobs must be processed on all machines with the same operation sequence. The flowshop scheduling problem occurs whenever it is necessary to schedule a set of n jobs on m machines in same order so that the result is optimal with respect to a specified objective. The great majority of published research on flowshop scheduling problem does not allow "job passing" due to the reduction in search space. The schedules that satisfy the assumptions are called permutation schedules. Permutation schedules are often applied in factories with conveyors between machines for material transfer and assembly lines performing the final assembly of bulky products [1].

There are $(n!)^m$ different schedules for ordering jobs on machines in non-permutation flowshop (NPFS), and the number of schedules for permutation flowshop (PFS) reduces to $n!$. When there are buffers between machines, good non-permutation schedules are likely to outperform their permutation counterparts. In other words, if the job sequences are allowed to be different on machines, the system performance will be improved significantly.

In most practical situation, firms often plan the production schedules to fulfill customer orders. However, lack of success in meeting the due dates will cause loss of the customers and loss of competitiveness. For the reasons, the tardiness criterion

* Corresponding author.
E-mail addresses: liao507@cjhuang.idv.tw (L.-M. Liao), cjhuang@cjhuang.idv.tw (C.-J. Huang).

0096-3003/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2010.05.089

is an important performance index. Therefore, this paper is concerned with minimizing the total tardiness on non-permutation flowshop scheduling problem.

Firstly, in this paper, three MILP models are constructed for NPFS problem with the objective of minimizing the total tardiness. Further, Tabu search based algorithms are developed to deal with large job-size problem for NPFS. Due to the exponential increase in search space with the number of machines (m), LH1 algorithm is developed to solve problem. As the number of machine is larger than 10, the paper also proposes LH2 algorithm by modifying the search structure of LH1 with counterbalancing the mass computation time consumed. Finally, the performances of the solved non-permutation schedules are evaluated.

The rest of this paper is organized as follows. In Section 2, the related works for NPFS are reviewed. Three MILP models for NPFS are constructed in Section 3. Section 4 proposes Tabu search based algorithms, denoted by LH1 and LH2, to deal with NPFS problems. Evaluate the improvement made by non-permutation schedules against permutation schedules with respect to total tardiness in Section 5. Finally, some conclusions are given and further studies are directed in Section 6.

2. Literature review

The most common criterion on flowshop scheduling problem is the minimization of makespan. A low value of makespan implies high utilization of machine. Ruiz and Maroto [2] gave a comprehensive review of many existing heuristics and meta-heuristics for permutation flowshop. The paper evaluated a total of 25 heuristics from classic Johnson's algorithm or dispatching rules to the most recent heuristics and advanced meta-heuristics. The comparisons were made against Taillard's [3] famous benchmark. Owing to the complexity of makespan calculation, the results were relatively hard to obtain. Furthermore, comparing to makespan criterion, the complexity of total completion time and due date related objectives was even harder [4]. But surveys of production scheduling show that meeting customer due dates is a critical concern for many manufacturing system [5]. Therefore, fulfilling customer orders is become prerequisite in modern industry.

Since the flowshop scheduling problems are NP-complete in most performance measures, it is unlikely to find an efficient method for solving the optimality of this kind of problem. Because of this, dynamic programming (DP) and branch and bound (B & B) approaches are applied to find an optimal schedule for the problem. Sonmez and Baykasoglu [6] developed a new DP formulation $n \times m$ job sequencing in a plastic pipe manufacturing facility (flowshop environment). In the formulation, setup times are considered and minimizing the total tardiness is the objective. Kim [1] developed a B & B algorithm for the m-machine permutation flowshop problem with the objective of minimizing the total tardiness. Afterward, Chung et al. [7] also proposed a B & B algorithm incorporating a machine-based lower bound and a dominance test for pruning nodes. The algorithm performs better than another B & B algorithm by Kim. The algorithm can handle job size up to 20. Pan et al. [8] devised dominance criteria to establish the precedence constraints between jobs in an optimal schedule for two-machine flowshop with the minimization of the total tardiness. Using the properties enhance the computational efficiency of the algorithm.

With regards to the flowshop problem with due date related objective, some of the works on the tardiness problem involve constructive heuristic and meta-heuristic. Kim [9] proposed several heuristics for minimizing the mean tardiness. It includes four constructive heuristics, GS2, NEHedd, NEHldd and ENS, and a meta-heuristic, Tabu search (TS). Later, Armentano and Ronconi [5] also proposed a heuristic based on TS technique with special strategies to solve the same problem in order to obtain better solutions in a reasonable time. Parthasarathy and Rajendran [10] applied simulated annealing (SA) to develop a heuristic for the mean tardiness and the weighted mean tardiness problem. The paper proposed a method for obtaining a good seed sequence and two perturbation schemes which are the random insertion perturbation scheme (RIPS) and curtailed random insertion perturbation scheme (CRIPS). The proposed SA based heuristic is better than the TS based one by Kim [9]. Afterward, Hasija and Rajendran [11] developed an improvement phase, job-index-based -insertion-scheme (JIBIS), to improve upon the seed solution. They also proposed two new perturbation schemes, job-shift-based (JSB) and probabilistic-step-swap (PSS) perturbation schemes, to the SA procedure for total tardiness problem. The experimental result verifies the superiority of the heuristic over the TS heuristic by Armentano and Ronconi [5] and SA heuristic by Parthasarathy and Rajendran [10]. In this paper, the comparisons of Armentano and Ronconi [5] and Hasija and Rajendran [11] against Taillard's [3] benchmark are also made, and the detailed comparisons are shown in Section 5.1.

Above papers are all considering permutation flowshop schedules (PFS) in flowshop and there are few papers taking account of non-permutation flowshop schedules (NPFS). Tandon et al. [12] compared the makespan values obtained by permutation and non-permutation schedules in flowshop. According to their computational experiments, the average percentage improvement of non-permutation schedule optimum over permutation schedule optimum is usually more than 1.5%. Koulamas [13] also proposed a simple constructive heuristic (HFC) to produce non-permutation schedules for the flowshop makespan problem. A flowline-based manufacturing system (FBMS) is a general flowshop which allows some jobs having missing operations on some machines. Pugazhendhi et al. [14] developed a heuristic to derive NPFS with makespan and flow time criteria and the effect on improving the schedule performance in FBMS is significant. Lin and Ying [15] proposed a hybrid approach, a combination of SA and TS, for the NPFS with the objective of minimizing the makespan.

Swaminathan et al. [16] examined three scheduling approaches for the flowshop tardiness problem. The third approach, "pure dispatching", represents the case where the flowshop's permutation requirement is completely relaxed and can obtain

non-permutation schedules. The computational experiments indicated that the ATC rule far outperforms the EDD, RANDOM and FIFO rules in all of the problem instances. Because "pure dispatching" rules are applied on individual machine, they are lack of global information pertaining to the flowshop and are myopic in nature. Therefore, the non-permutation schedule by ATC rule is not sure better than good permutation schedule. Liao et al. [17] also compared the makespan, total tardiness and total weighted tardiness values obtained by permutation and non-permutation schedules. The computational results show that the percentage improvement is quite small (less than 0.5%) with respect to makespan, but it is significant (in the range of 1–5%) with respect to the total tardiness, $\sum T_i$, and the weighted total tardiness, $\sum w_i T_i$. More importantly, the maximum percentage improvements in many cases of $\sum T_i$ and $\sum w_i T_i$ are larger than 10%. The results provide a future research direction for flowshop non-permutation flowshop problem. Therefore, this paper develops heuristics based on TS technique for the non-permutation flowshop with total tardiness problem.

3. Notations and MILP models formulation

Stafford and Tseng [18] proposed a Stafford's model for permutation flowshop (PFS) problem. Tseng et al. [19] assessed the relative effectiveness of four mixed-integer linear programming (MILP) models for the PFS with minimizing makespan problem. The models can be divided into two classes, one is assignment problem based models and the other is dichotomous constraints based models. Later, Stafford et al. [20] investigated the performance of two families of mixed-integer linear programming (MILP) models for solving the regular permutation flowshop problem to minimize makespan. The Wagner's and Wilson's models [21,22] used the classic assignment problem. The Manne model [23] used pairs dichotomous constraints to control the assignment of jobs to various sequence positions [19]. The Liao–You model [24] combined each pair of inequality dichotomous constraints from the original Manne model into a single equality constraint. The constraints set is equal to a surplus variable which account precedence relationship of two jobs on each machine. This paper extends the MILP models for the PFS with minimizing makespan problem [19,20] to formulate three MILP models for the NPFS with total tardiness problem. Model 1 is the assignment problem based model which refers to Wilson's model. Models 2 and 3 are dichotomous constraints based models which refer Manne's and Liao–You's models, respectively.

The notations used in this paper are defined as follows.

Parameters

$P_{r,i}$ = processing time of job i on machine r

d_i = the due date of job i

M = a sufficiently large positive value (mathematically, $M \to \infty$)

Decision variables

$Y_{r,i,j}$ = 1, if job i precedes job j on machine r; 0 otherwise

$Z_{r,i,l}$ = 1, if job i is assigned to sequence position l on machine r; 0 otherwise

Dependent variables

$B_{r,l}$ = start (begin) time of job in sequence position l on machine r

$C_{r,i}$ = completion time of job i on machine r

dd_l = due date of the job in position l

$T_{[l]}$ = tardiness of the job in position l

T_i = tardiness of job i

The subscript symbols are: r for machines, $r = 1,2,\ldots,m$; i and j for jobs, $i,j = 1,2,\ldots,n$; and l and l' for the sequence positions, $l,l' = 1,2,\ldots,n$, where the parameter m and n represent the number of machines and jobs, respectively. In Model 3, $q_{r,i,j}$ is a surplus variable. The three proposed MILP models as follows.

Model 1.

$$\text{Minimize} \quad \sum_{l=1}^{n} T_{[l]}, \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} Z_{r,i,l} = 1 \quad r = 1,2,\ldots,m; \quad l = 1,2,\ldots,n, \tag{2}$$

$$\sum_{l=1}^{n} Z_{r,i,l} = 1 \quad r = 1,2,\ldots,m; \quad i = 1,2,\ldots,n, \tag{3}$$

$$B_{1,1} = 0, \tag{4}$$

$$B_{r,1} + \sum_{i=1}^{n} p_{r,i} Z_{r,i,1} \leqslant B_{r+1,1} \quad r = 1,2,\ldots,m-1, \tag{5}$$

$$B_{r,l} + u m_{i=1}^{n} p_{r,i} Z_{r,i,l} \leqslant B_{r,l+1} \quad r = 1,2,\ldots,m; \quad l = 1,2,\ldots,n-1, \tag{6}$$

$$(2 - Z_{r,i,l} - Z_{r+1,i,l'})M \geqslant B_{r,l} + p_{r,i} - B_{r+1,l'} \quad r = 1,2,\ldots,m-1; \quad i,l,l' = 1,2,\ldots,n, \tag{7}$$

$$dd_l \leqslant d_i \times (1 - Z_{m,i,l}) \times M + d_i \quad i,l = 1,2,\ldots,n, \tag{8}$$

$$T_{[l]} \geqslant B_{m,l} + \sum_{i=1}^{n} p_{mi} Z_{m,i,l} - dd_l \quad l = 1,2,\ldots,n, \tag{9}$$

$$B_{r,l} \geqslant 0, \quad T_{[l]} \geqslant 0, \quad dd_l \geqslant 0, \quad Z_{r,i,l} = 0 \text{ or } 1 \quad r = 1,2,\ldots,m; \quad i,l = 1,2,\ldots,n. \tag{10}$$

Eq. (1) represents the objective function which seeks to minimize the sum of tardiness over all jobs. Constraint (2) ensures that each job is assigned to exactly one position of job sequence on every machine. Constraint (3) states that each position of job sequence processes exactly one job on every machine. Constraints (4) and (5) denote the starting times of the first job on every machine. Constraint (6) insures that the $(l+1)$th job in the sequence of machine r does not start on machine r until the lth job in the sequence of machine r has completed. Constraint (7) insures that the starting time of jobs i which is assigned to position l in the sequence on machine $r+1$ is not earlier than its finish on machine r. Constraint (8) is used to compute the due date of the lth job in the sequence of machine m, where M is defined as a very large constant. Constraints (9) is used to compute the tardiness of the lth job in the sequence of machine m. Constraint (10) imposes the binary and non-negative restrictions on the variables.

Model 2.

$$\text{Minimize} \quad \sum_{i=1}^{n} T_i, \tag{11}$$

$$\text{subject to} \quad C_{1,i} \geqslant p_{1,i} \quad i = 1,2,\ldots,n, \tag{12}$$

$$C_{r+1,i} - C_{r,i} \geqslant p_{r+1,i} \quad r = 1,2,\ldots,m-1; \quad i = 1,2,\ldots,n, \tag{13}$$

$$C_{r,i} - C_{r,j} + MY_{r,i,j} \geqslant p_{r,i} \quad r = 1,2,\ldots,m; \quad i,j = 1,2,\ldots,n \quad \text{and} \quad i < j, \tag{14}$$

$$C_{r,i} - C_{r,j} + MY_{r,i,j} \leqslant M - p_{r,j} \quad r = 1,2,\ldots,m; \quad i,j = 1,2,\ldots,n \quad \text{and} \quad i < j, \tag{15}$$

$$T_i \geqslant C_{m,i} - d_i \quad i = 1,2,\ldots,n, \tag{16}$$

$$C_{r,i} \geqslant 0, \quad T_i \geqslant 0, \quad Y_{r,i,j} = 0 \text{ or } 1 \quad r = 1,2,\ldots,m; \; i,j = 1,2,\ldots,n \quad \text{and} \quad i < j. \tag{17}$$

Constraint (12) ensures that the completion time of each job on machine 1 is not earlier than duration of the job's processing time on machine 1. Constraint (13) ensures that each job's completion time on machine $r+1$ is not earlier than the job's completion time on machine r. Disjunctive constraints (14) and (15) establish the relationship between the completion time of jobs i and j. They insure the job i either precedes job j or follows job j in the sequence, but not both. Constraint (16) is used to compute the tardiness of job i. Constraint (17) imposes the binary and non-negative restrictions on the variables.

Model 3.

$$\text{Minimize} \quad \sum_{i=1}^{n} T_i, \tag{18}$$

$$\text{subject to} \quad C_{1,i} \geqslant p_{1,i} \quad i = 1,2,\ldots,n, \tag{19}$$

$$C_{r+1,i} - C_{r,i} \geqslant p_{r+1,i} \quad r = 1,2,\ldots,m-1; \quad i = 1,2,\ldots,n, \tag{20}$$

$$C_{r,i} - C_{r,j} + MY_{r,i,j} - p_{r,i} = q_{r,i,j} \quad r = 1,2,\ldots,m; \quad i,j = 1,2,\ldots,n \quad \text{and} \quad i < j, \tag{21}$$

$$q_{r,i,j} \leqslant M - p_{r,i} - p_{r,j} \quad r = 1,2,\ldots,m; \quad i,j = 1,2,\ldots,n \text{ and } i < j, \tag{22}$$

$$T_i \geqslant C_{m,i} - d_i \quad i = 1,2,\ldots,n, \tag{23}$$

$$C_{r,i} \geqslant 0, \quad T_i \geqslant 0, \quad q_{r,i,j}, \quad Y_{r,i,j} = 0 \text{ or } 1 \quad r = 1,2,\ldots,m; \quad i,j = 1,2,\ldots,n \quad \text{and} \quad i < j. \tag{24}$$

Liao–You model combines each pair of inequality dichotomous constraints from the Manne model into a single equality constraint. The model sets a surplus variable, $q_{r,i,j}$, which accounts for the precedence relationship of job i and job j on machine r. The disjunctive constraints (21) and (22) establish the relationship between the completion time of jobs i and j by adding a boundary constraint on surplus variable $q_{r,i,j}$. Constraints (19), (20), (23) and (24) are the same as constraints (12), (13), (16) and (17) of Model 2.

Table 1 illustrates the complexity of the three MILP models according to the number of constraints and binary variables. Obviously, the number of constraints and binary variables in Model 1 is more complicated than Models 2 and 3.

The above three MILP models were solved by LINGO 10.0 and run on a Pentium IV 1.8 GHz (Dual Core) PC. The models can only handle test problems with machine's size 5 and job's size up to 8,$n \leqslant 8$, in 12 h. The resultant computer solution time for the superiority of MILP model is revealed as follows: Model 3 is the best MILP model and Model 2 is the second best one. That is, Models 2 and 3 which are both based on dichotomous constraints are more efficient than Model 1 which is based on assignment problem.

Table 1
The number of constraints and binary variables for MILP models.

Model	Number of constraints	Number of binary variables
1	$n^3(m-1)+3mn+n^2+n$	n^2m
2	mn^2+n	$mn(n-1)/2$
3	mn^2+n	$mn(n-1)/2$

4. Proposed algorithms

In this section, TS technique is adopted to develop heuristics for the flowshop total tardiness problems to produce non-permutation schedules. In order to obtain an initial solution, we compare two recently developed permutation schedule algorithms by Armentano and Ronconi [5] and Hasija and Rajendran [11]. The Armentano and Ronconi proposed a basic TS structure with an initial solution generated by LB_NEH algorithm. Their heuristic is named as AR_TS algorithm in the current study. The Hasija and Rajendran proposed a SA procedure with two new perturbation schemes, i.e. job-shift-based (JSB) and probabilistic-step-swap (PSS) perturbation schemes. The heuristic is named as HR_SA algorithm in this study. The procedures of the AR_TS and HR_SA algorithms are respectively described in Appendix.

The proposed heuristics, based on the TS technique for the flowshop total tardiness problems, are designed to produce non-permutation schedules and named as LH1 and LH2 algorithms respectively. The LH1 and LH2 algorithms are proposed to solve $m \leqslant 10$ and $10 < m \leqslant 20$ NPFS problems, respectively.

In the meta-heuristic, the initial solution is an important parameter which affects the final solution's performance. This paper, therefore, selects the best initial permutation schedule from AR_TS and HR_SA as initial solution. From the comparisons of the two algorithms, the performances of the solution obtained by AR_TS are better than HR_SA. The detailed comparisons are shown on Table 4 in Section 5.1. The procedure of LH1 algorithm is described as follows.

Step 1. Initial solution: The initial sequence is obtained by AR_TS algorithm.

Step 2. Neighborhood structure: Define $J_{r,[l]}$ as the lth job on machine r, and the two adjacent jobs, $J_{r,[l]}$ and $J_{r,[l+1]}$, are selected as the lth job-pair. Therefore, there are $n-1$ job-pairs on machine r where r is from 1 to m. And then try to swap the jobs in the lth job-pairs on each machine respectively. Neighborhoods are formed with at least one job-pair in which the jobs were swapped. Consequently, there are (2^m-1) neighborhoods at the lth job-pair and the total size of neighborhood is $(2^m-1)\times(n-1)$. The structure of neighborhoods is described in Fig. 1.

Step 3. Selection of the best neighborhood: Given a solution, each neighborhood solution is evaluated and the best solution is selected.

Step 4. Tabu list: In order to record job sequences on each machine, a virtual job sequence is created by linking job sequences on sequential machines. Hashing function is applied to create a Tabu list to record all job sequences. The hashing value can be obtained from the following formula [25,26]:

$$h=\left[\sum_{l=1}^{mn} z(x_l)\times z(x_{l+1})\right] \bmod [\mathrm{MAXINT}+1] \tag{25}$$

where x_l is the job in position l of virtual job sequence; $z(x_l)$ is an integer drawn from the range (1,2,...,32,767), MAXINT = 65,535 and $x_{mn+1}=x_1$, where m and n are number of machine and job, respectively.

Step 5. Tabu tenure: By preliminary experiment analysis, the size of Tabu tenure is not proved to have significant effect on objective value, makespan. Therefore, magic number 7 is selected as the size of Tabu tenure.

Step 6. Stopping criterion: The stopping criterion is 10 successive solutions with no improvement, then stops searching procedure. The average operation time of CPU used for each problem size is presented in Table 5.

The LH2 algorithm is proposed to solve $10 < m \leqslant 20$ NPFS problems. The procedure is similar to that of LH1 except for Step 2. The main idea of Step 2 of LH2 is making effort to decrease the number of neighborhoods. As the number of machines is m, the number of neighborhoods is $(2^m-1)\times(n-1)$. The number of neighborhoods increases exponentially with m. Consequently, all machines are divided into two machine groups to decrease the number of neighborhoods. The first machine group consists of machines 1,2,...,m_c, and the second machine group consists of machines $m_c+1, m_c+2, \ldots, m$. Preliminary experiments reveal that the variable number of machines will obtains more good solutions than fixed ones. Taking time and solution quality into account, the numbers of machine groups are ranged from $\lfloor 0.35\ m \rfloor$ to $\lfloor 0.65\ m \rfloor$, where $\lfloor x \rfloor$ is the largest integer not larger than x. The Step 2 of LH2 algorithm is described as follows.

Step 2. All machines are divided dynamically into two machine groups. The number of machines in each group is generated from U($\lfloor 0.35\ m \rfloor$,$\lfloor 0.65\ m \rfloor$). The first machine group consists of machines 1,2,...,m_c and the remaining machines belong to the second machine group. Execute Step 2 of LH1 on two machine groups respectively. And then combine the neighborhoods of each machine group. The size of neighborhood is not larger than $3\times(n-1)(2^{\lfloor 0.65\ m \rfloor}+2^{\lfloor 0.35\ m \rfloor}-2)$. The structure of neighborhoods is described in Fig. 2.

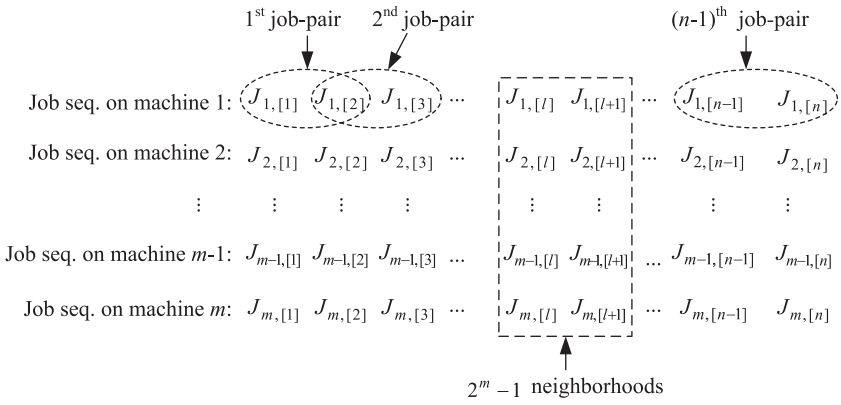

Fig. 1. Neighborhood structure of LH1.

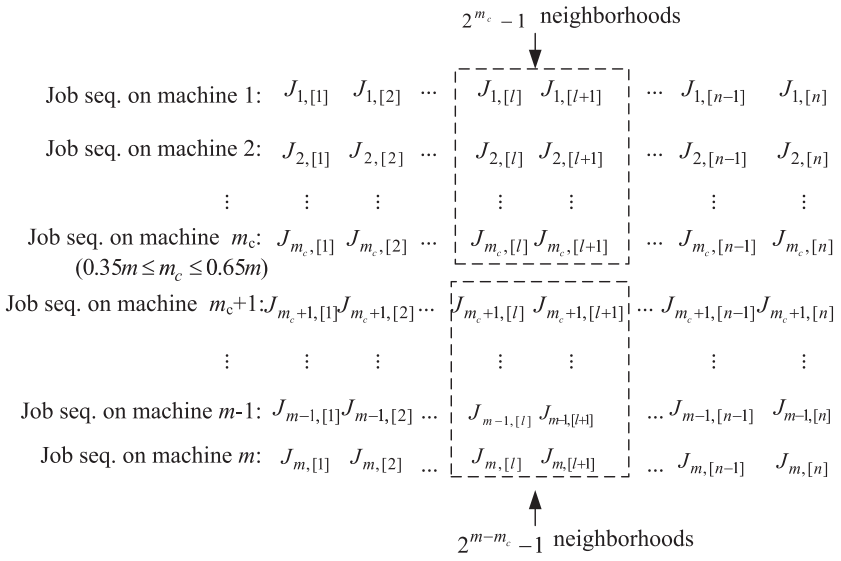

Fig. 2. Neighborhood structure of LH2.

5. Computational results

This section presents the evaluation of the performances of LH1 and LH2 algorithms through computational experiments, where all the algorithms were coded in the Visual Basic programming language and run on a Pentium IV 1.8 GHz (Dual Core) PC. The test problems chosen for evaluating the algorithms are the benchmark problems by Taillard [3]. The total numbers of jobs, n, are 20, 50 and 100 while the numbers of machines, m, are 5, 10 and 20. There are 9 combinations of $n \times m$ under four different scenarios of due dates, thus resulting in 36 problem combinations ($9 \times 4 = 36$). The four different scenarios of due dates are generated between $P(1 - T - R/2)$ and $P(1 - T + R/2)$, where T and R are the tardiness factors of jobs and dispersion ranges of due dates, respectively [5]. P is defined as

$$P = \max\left\{\max_{1\leq r\leq m}\left\{\sum_{i=1}^{n} p_{ri} + \min\sum_{l=1}^{r-1} p_{ri} + \min\sum_{l=r+1}^{m} p_{ri}\right\}, \max\sum_{r=1}^{m} p_{ri}\right\}, \tag{26}$$

where p_{ri} is the processing time of job i on machine r. In the following are the four different scenarios configured by varying T and R.

Scenario I: Low tardiness factor (T = 0.2) and small due date range (R = 0.6)
Scenario II: Low tardiness factor (T = 0.2) and wide due date range (R = 1.2)
Scenario III: High tardiness factor (T = 0.4) and small due date range (R = 0.6)
Scenario IV: High tardiness factor (T = 0.4) and wide due date range (R = 1.2)

Each problem combination runs 10 replications in the four different scenarios of due dates. The total runs are 360.

Table 2
Computational time of HR_SA algorithm.

Size $m \times n$[a]	Average computational time (s)				Size $m \times n$[a]	Average computational time (s)				Size $m \times n$[a]	Average computational time (s)			
	I	II	III	IV		I	II	III	IV		I	II	III	IV
5×20	<1[b]	<1[b]	<1[b]	<1[b]	10×20	<1[b]	<1[b]	<1[b]	<1[b]	20×20	1	1	1	1
5×50	3	2	8	9	10×50	9	10	13	13	20×50	17	18	18	18
5×100	17	18	95	104	10×100	23	25	132	139	20×100	165	149	177	184

[a] $m \times n$ denotes (the number of machines) $\times$ (the number of jobs).
[b] The computational time is less than one second.

Table 3
Computational time of AR_TS algorithm.

Size $m \times n$[a]	Average computational time (s)				Size $m \times n$[a]	Average computational time (s)				Size $m \times n$[a]	Average computational time (s)			
	I	II	III	IV		I	II	III	IV		I	II	III	IV
5×20	<1[b]	<1[b]	<1[b]	<1[b]	10×20	<1[b]	1	<1[b]	<1[b]	20×20	1	2	1	1
5×50	1	1	3	2	10×50	34	20	22	20	20×50	12	21	22	21
5×100	<1[b]	<1[b]	85	59	10×100	4	1	97	106	20×100	141	59	178	147

[a] $m \times n$ denotes (the number of machines) $\times$ (the number of jobs).
[b] The computational time is less than one second.

Table 4
Relative percentage decrease of AR_TS and HR_SA algorithms.

Size $m \times n$	Mean of RPD_{AR_TS} (%)				Mean of RPD_{HR_SA} (%)			
	I	II	III	IV	I	II	III	IV
5×20	25.73	2.79	7.38	4.63	0	0	0	0
5×50	23.45	3.47	6.21	8.45	0	0	0.81	0
5×100	–[a]	–[a]	7.41	9.66	–[a]	–[a]	0	0
10×20	10.83	1.82	4.36	3.43	0.06	0	0	0.08
10×50	53.54	25.26	8.81	5.87	0	0	0	0
10×100	–[a]	–[a]	7.48	7.02	–[a]	–[a]	0.18	0.22
20×20	4.05	4.19	2.35	3.03	0.02	0	0	0
20×50	14.96	9.67	6.62	4.48	0	0	0	0
20×100	32.86	14.67	9.33	7.32	0.14	0	0	0

[a] None instance of PFS problems with $\sum T > 0$.

5.1. Initial solution selected

In the meta-heuristic, the initial solution is an important parameter which affects the final solution performance. Accordingly, the two relative percentage decrease, RPD_{AR_TS} and RPD_{HR_SA}, are used to evaluate the AR_TS and HR_SA algorithms respectively. The formulas are described as follows.

$$RPD_{AR\_TS} = \frac{\mathrm{Max}\{TT_{AR\_TS}, TT_{HR\_SA}\} - TT_{AR\_TS}}{\mathrm{Max}\{TT_{AR\_TS}, TT_{HR\_SA}\}} \times 100\%, \tag{27}$$

$$RPD_{HR\_SA} = \frac{\mathrm{Max}\{TT_{AR\_TS}, TT_{HR\_SA}\} - TT_{HR\_SA}}{\mathrm{Max}\{TT_{AR\_TS}, TT_{HR\_SA}\}} \times 100\%, \tag{28}$$

where TT_{AR_TS} and TT_{HR_SA} denote the total tardiness obtained by AR_TS and HR_SA algorithms, respectively.

According to Hasija and Rajendran [11], this paper executes six problem sizes and obtains their average computational time as shown in Table 2. Similarly, Table 3 describes the average computational time of the AR_TS algorithm according to Armentano and Ronconi [5]. The computational results of the AR_TS and HR_SA algorithms are presented in Table 4. It is evident that the AR_TS algorithm is better than HR_SA. Therefore, the AR_TS algorithm is selected as the initial solution of the proposed algorithms in this paper.

5.2. LH1 and LH2 algorithms performance

Liao et al. [17] compared the total tardiness values obtained by permutation and non-permutation schedules. The computational results showed that the percentage improvement is significant (in the range of 1–5%) with respect to the total tardiness, $\sum T_i$. Even the maximum percentage improvements in most of the cases in $\sum T_i$ are larger than 10%. To evaluate the LH1 and LH2 algorithms, these two algorithms are first compared with the Tabu search algorithm proposed by Liao et al., which is denoted as L_TS in this current paper. The procedures of the L_TS is described in Appendix.

Liao et al. [17] deem that the sequence of machine r affects the right successive machine $r+1$. That is, it is very likely that the sequences of two adjacent machines are the same. Therefore, in determining the sequence on machine $r+1$, the sequence on the succeeding machine is the same as that on machine r. It deserves to be mentioned that the L_TS algorithm only consumes less time. The proposed algorithms, LH1 and LH2, are significantly superior to the L_TS algorithm, but they consume more time. The comparison of performances is shown in Tables 5 and 6.

Further, the proposed algorithms, LH1 and LH2, are analyzed in terms of solution quality. LH1 and LH2 are evaluated by relative percentage improvement (PI) of non-permutation schedules with respect to the AR_TS algorithm. The average and maximum percentage improvements are described as follows.

(1) Average percentage improvement:

$$API = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{TT_{AR\_TS}(i) - TT_{LH}(i)}{TT_{AR\_TS}(i)}\right) \times 100\%. \tag{29}$$

(2) Maximum percentage improvement:

$$MPI = \max_{i}\left(\frac{TT_{AR\_TS}(i) - TT_{LH}(i)}{TT_{AR\_TS}(i)}\right) \times 100\%, \tag{30}$$

where TT_{AR_TS} and TT_{LH} are the total tardiness by the AR_TS and LH1 (or LH2) algorithms, respectively.

Tables 7 and 8 present the results of the percentage improvement of the proposed LH1 and LH2 algorithms respectively. The total tardiness permutation schedules in 181 out of the 240 instances are larger than zero, as shown in Table 7. In the 181 instances, 156 cases are improved by the LH1 algorithm. As the numbers of machines are 5 and 10, the percentages of improved instances are about 77.1% and 98.5%, respectively. Similarly, Table 8 shows that the total tardiness permutation schedules in 118 out of the 120 instances are larger than zero. In the 118 instances, 116 cases are improved by the LH2 algorithm. As the number of machines is 20, the percentage of improved instances is more than 98.3%. The results show that the non-permutation schedules are better than the permutation schedules for flowshop tardiness problems.

Furthermore, the average percentage improvement (API) value is used to evaluate the solution quality of LH1 and LH2, and the results are shown in Tables 9 and 10. As $m = 5$ and 10, API and MPI values are 6.06% and 100.00%, respectively. As $m = 20$, API and MPI values are 9.45% and 94.29%, respectively. LH1 and LH2 are significant improvement under 4 different scenarios of due dates. Scenario I (T = 0.2 and R = 0.6) particularly presents the most improvement, where the API values are 15.65% and 15.47% for $m = 5,10$ and $m = 20$ problems, respectively. The second greatest improvement is with Scenario II (T = 0.2 and R = 1.2), where the API values are 12.14% and 14.66% for $m = 5,10$ and $m = 20$ problems, respectively. It can be concluded that exploring the non-permutation schedule is effective and necessary for low tardiness factor flowshop tardiness problems. More importantly, the MPI are 100% in both Scenarios I and II in Table 9. In Scenarios I of Tables 7 and 8, there are 62 instances with $\sum T > 0$ and 34 instances with $PI \geqslant 5\%$. Similarly, in Scenarios II of Tables 7 and 8, there are 57 instances with $\sum T > 0$ and 28 instances with $PI \geqslant 5\%$. The ratios of $PI \geqslant 5\%$ are about 55% and 49% in Scenarios I and II, respectively. Furthermore, there are 24 instances $PI \geqslant 10\%$ with and the ratio is about 39% in Scenarios I, and 14 instances $PI \geqslant 10\%$ with and the ratio is about 25% in Scenarios II.

On the other hand, it is observed in Tables 9 and 10 that as the number of jobs increases, the percentage improvement is also raised. Moreover, as the problems are with more jobs, the sequence of jobs becomes more flexible. It is also noticeable in Table 9 that as the number of machines increases, the percentage improvement is raised. However, the API of $m = 20$ in Table 10 is slightly worse than the API of $m = 10$ in Table 9. The main reason is that there are too many neighborhoods of moves $m = 20$ to effectively search for better solutions in a reasonable time. In order to effectively search for another better solution, the searching strategy must reduce the size of neighborhood (refer to step 2 of LH2). But as the neighborhoods are reduced, some better solutions may not be explored. Therefore, the percentage improvement of $m = 20$ by LH2 is slightly worse than the one of $m = 10$ by LH1.

Table 5
Relative performance of L_TS and LH1 algorithms.

Size $m \times n$	Mean ratio of $TT_{L\_TS}/TT_{LH1} \times 100$[a]				Average computational time of LH1 (s)			
	I	II	III	IV	I	II	III	IV
5×20	102.22	102.59	100.21	100.54	<1[b]	<1[b]	<1[b]	<1[b]
5×50	–[c]	101.04	100.07	100.44	<1[b]	1	<1[b]	<1[b]
5×100	–[c]	–[c]	100.10	100.09	<1[b]	<1[b]	<1[b]	<1[b]
10×20	102.20	101.00	100.92	100.67	<1[b]	<1[b]	<1[b]	<1[b]
10×50	157.48	168.30	100.83	100.88	2	3	14	12
10×100	–[c]	–[c]	100.38	100.42	<1[b]	<1[b]	63	59

[a] Mean ratio of permutation schedule (PS) with $\sum T > 0$.
[b] The computational time is less than one second.
[c] Instance of permutation schedule (PS) with $\sum T > 0$ is zero.

Table 6
Relative performance of L_TS and LH2 algorithms.

Size $m \times n$	Mean ratio of $TT_{L\_TS}/TT_{LH2} \times 100$[a]				Average computational time of LH2 (s)			
	I	II	III	IV	I	II	III	IV
20×20	101.40	101.97	100.66	100.37	14	15	17	16
20×50	104.48	102.98	101.44	101.55	281	237	409	369
20×100	136.82	330.92	100.95	100.63	1256	701	2008	1913

[a] Mean ratio of permutation schedule (PS) with $\sum T > 0$.

Table 7
Percentage of improved instances by LH1 algorithm.

Size $m \times n$	Instances of PFS with $\sum T > 0$				Number of improved instances				Improved instances (%)
	I	II	III	IV	I	II	III	IV	
5×20	10	10	10	10	5	4	5	8	55.0
5×50	2	1	10	10	2	1	9	9	91.3
5×100	0	0	10	10	0	0	10	7	85.0
Average									77.1
10×20	10	10	10	10	10	9	10	10	97.5
10×50	10	7	10	10	10	6	10	10	97.9
10×100	0	1	10	10	0	1	10	10	100.0
Average									98.5

Table 8
Percentage of improved instances by LH2 algorithm.

Size $m \times n$	Instances of PFS with $\sum T > 0$				Number of improved instances				Improved instances (%)
	I	II	III	IV	I	II	III	IV	
20×20	10	10	10	10	10	10	10	10	100.0
20×50	10	10	10	10	10	9	10	10	97.5
20×100	10	8	10	10	10	7	10	10	97.4
Average									98.3

Table 9
Improvement percentage of LH1.

Size $m \times n$	API (%)				Average	MPI (%)			
	I	II	III	IV		I	II	III	IV
5×20	5.94	2.74	1.02	1.61	2.66	18.69	9.01	17.37	16.66
5×50	51.46	1.03	0.53	3.48	6.26[b]	100.00	1.03	1.78	3.79
5×100	–[a]	–[a]	0.72	2.24	1.48	–[a]	–[a]	1.28	7.69
10×20	5.41	4.63	2.79	1.97	3.70	11.34	14.73	8.48	9.36
10×50	28.43	25.33	3.43	2.65	14.12	37.84	81.82	7.33	6.52
10×100	–[a]	100.00	2.20	2.53	7.01	–[a]	100.00	3.90	6.22
Average	15.65	12.14	1.78	2.41	6.06[c]				

[a] Instance of permutation schedule (PS) with $\sum T > 0$ is zero.
[b] $6.26 \approx (51.46 \times 2 + 1.03 \times 1 + 0.53 \times 10 + 3.48 \times 10) \div 23$.
[c] $6.06 \approx (2.66 \times 40 + 6.26 \times 23 + 1.48 \times 20 + 3.70 \times 40 + 14.12 \times 37 + 7.01 \times 21) \div 181$.

Table 10
Improvement percentage of LH2.

Size $m \times n$	API (%)				Average	MPI (%)			
	I	II	III	IV		I	II	III	IV
20×20	4.32	4.19	1.83	1.45	2.95	8.27	8.71	2.18	2.90
20×50	18.67	11.16	7.15	4.77	10.44	16.72	15.07	9.27	8.28
20×100	23.42	32.11	2.86	3.14	15.24	74.07	94.29	4.85	5.46
Average	15.47	14.66	3.95	3.12	9.45				

6. Conclusions

This paper has addressed the non-permutation flowshop scheduling (NPFS) problem with the objective of minimization total tardiness. Three mix integer programming (MILP) models have been constructed and the resultant computer solution time has been analyzed for the superiority of the MILP model. The result indicates that the dichotomous constraints based model is superior to the assignment problem based model. Specifically, Model 3 which refers to Liao–You model is the best. The optimal non-permutation schedule can be found by the MILP model for small problems with $n \leqslant 8$ within 24 h. The job sequences of two adjacent machines are similar, and the position of each job in a given sequence can't be drastically varied. So, two adjacent jobs swap is applied to create the neighborhood of job sequence. Due to the fact that local search often traps into local optimal, the Tabu search technique is applied to the proposed LH1 and LH2 algorithms to effectively solve the NPFS problems with $m \leqslant 20$. The algorithms aim to guide the search by exploring the solution space of a problem beyond local optimality. The computational results show that the percentage of improved instances by LH1 and LH2 algorithms is about 87.8% (77.1% + 98.5%)/2 and 98.3%, respectively. As the number of machines is equal to 10 or 20, the percentage of improved instances is more than 98%. The results also show that the non-permutation schedules are better than the permutation schedules for flowshop tardiness problems.

The non-permutation schedules have achieved significant improvement in different scenarios. Scenario I of due date (T = 0.2 and R = 0.6) especially presents the greatest improvement, where the API is 15.56%, (15.65%×32 + 15.4%×30)/62. Another improvement is achieved with Scenario II of due date (T = 0.2 and R = 1.2), where the API is 13.38%, (12.14% × 29 + 14.66% × 28)/57. Consequently, API is equal to 14.52%, (15.56% × 62 + 13.38% × 57)/119, for flowshop problems with low tardiness factors. It is evident that exploring non-permutation schedule is effective and necessary for low tardiness factors.

As $m = 20$, the proposed LH2 algorithm divides all machines into two machine groups to decrease the size of neighborhood. Due to the decreased neighborhood size some good non-permutation schedules may not be found, but in this paper, it can be improved by using dynamical machine grouping. Accordingly, how to design the neighborhood of the move is a crucial topic for further studies. In addition, constructing another meta-heuristic to find good non-permutation schedules is a further research direction.

Acknowledgement

This work was supported by funding from the National Science Council of the Republic of China, and special thanks to all who have helped to make this study.

Appendix A. AR_TS algorithm

The procedure of the AR_TS algorithm is first introduced in the following steps [5].

Step 1. Initial solution: Jobs are ordered in a non-decreasing order in $I_i = d_i - \sum_r^m p_{ri}$. The NEH algorithm is associated with the I_i dispatching rule. That is, jobs are sorted by the I_i rule. Then a final solution is obtained in a constructive way, adding in each step a new job in the order which is then inserted in the best place.
Step 2. Neighborhood structure: Let $\pi(i)$ denote the position of job i located in a given sequence. A job i, located at $\pi(i)$, is inserted at position y arbitrary; this move generates a neighborhood of size $(n-1)^2$.
Step 3. Selection of the best neighborhood: Given a solution, each neighborhood solution is evaluated and the best solution is selected.
Step 4. Tabu list: Job i selected for insertion is the attribute of Tabu list.
Step 5. Tabu tenure: The duration of the restriction rule applied to move is set dynamically. For every 20 iterations a value for Tabu tenure is generated from an uniform distribution between $n/2$ and n, and at each iteration this value is adjusted. If the value of selected solution is larger than the current solution, the Tabu tenure associated with the selected solution is increased by one unit. If the value of selected solution is smaller than the current one, the Tabu tenure is decreased by one unit.
Step 6. Aspiration criterion: A move can be released from the Tabu condition if it has a solution value better than the best solution found by the search.
Step 7. Stopping criterion: The stopping criterion is a certain amount of CPU time.

Appendix B. HR_SA algorithm

The procedure of the HR_SA algorithms is described as follows [11].

Step 1. Initial solution: Jobs are ordered in ascending order in d'_{ri} for every machine r, where $d'_{1i} = (d_i \times p_{1i})/\sum_{r=1}^m p_{ri}$ and $d'_{ri} = d'_{r-1,i} + (d_i \times p_{ri})/\sum_{k=1}^m p_{ki},\ r = 2, 3, \ldots, m$. Hence get m sequences corresponding to m machines. Select the sequence with the least value of total tardiness as the seed sequence. Assign this sequence to both S and B, where S is the sequence on which JSB and PSS perturbation schemes are implemented, and B is the best sequence obtained so far.

Step 2. The seed sequence is performed the improvement scheme by job-index-based insertion-scheme (JIBIS) on S and B.
Step 3. Set initial temperature $T = 540$.
Step 4. 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 5. Execute JSB perturbation $2n$ times and store an archive of best ten sequences, S'.
Step 6. Execute PSS perturbation twice and still store an archive of best ten sequences, S'.
Step 7. If S' is better than S, set $S = S'$ (replace), otherwise, set $S = S'$ with a probability of $\exp\{[-(Z' - Z) \times Z]/T\}$, where Z and Z' are the total tardiness of S and S', respectively.
Step 8. Stopping criterion: If there are continuous $2n$ times not improvement, then calculate the replace ratio, number of replace $\times$ 100/number of move. If the ratio is less than or equal to 15, then "FREEZE" counter is added by 1, otherwise, set $T = 0.9 \times T$, where 0.9 is the reduction factor. If FREEZE = 5 or $T \leqslant 20$, then implement the improvement scheme JIBIS on all sequences stored in the archive. The best sequence of all the sequences generated by JIBIS is returned as the heuristic solution.

In the meta-heuristic, the initial solution is an important parameter which affects the final solution performance. In order to obtain a good initial solution for non-permutation flowshop problem, this paper select the better one of AR_TS and HR_SA algorithms. The relative percentage decrease, RPD_i, is used to evaluate the two algorithms. The results show that AR_TS performed better than HR_SA. The detailed description is illustrated in Section 5.1.

Appendix C. L_TS algorithm

The procedure of the algorithm is described as follows [17].

Step 1. Machine $r = 1,2$. The sequences on machines 1 and 2 are the same as the permutation schedule.
Step 2. Machine $r = 3,4,\ldots,m$. Using the sequence machine $r - 1$ as the initial solution, perform the short term search of TS to obtain the sequence on machine r. In determining the sequence on machine r, the sequences on preceding machines are fixed, while the sequences on the successive machines are the same as those on machine r.

References

[1] Y.D. Kim, Minimizing total tardiness in permutation flowshops, European Journal of Operational Research 85 (1995) 541–555.
[2] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2005) 479–494.
[3] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278–285.
[4] M. Pinedo, Scheduling: Theory, Algorithms and Systems, second ed., Prentice-Hall, Englewood Cliffs, NJ, 2002.
[5] V.A. Armentano, D.P. Ronconi, Tabu search for total tardiness minimization in flowshop scheduling problems, Computers and Operations Research 26 (1999) 219–235.
[6] A.I. Sonmez, A. Baykasoglu, A new dynamic programming formulation of $(n \times m)$ flowshop sequencing problems with due dates, International Journal of Production Research 36 (1998) 2269–2283.
[7] C.S. Chung, J. Flynn, O. Kirca, A branch and bound algorithm to minimize the total tardiness for m-machine permutation flowshop problems, European Journal of Operational Research 174 (2006) 1–10.
[8] J.C.H. Pan, J.S. Chen, C.M. Chao, Minimizing tardiness in a two-machine flowshop, Computers and Operations Research 29 (2002) 869–885.
[9] Y.D. Kim, Heuristics for flowshop scheduling problems minimizing mean tardiness, Journal of Operational Research Society 44 (1993) 19–28.
[10] S. Parthasarathy, C. Rajendran, Simulated annealing approach to scheduling in a flowshop with multiple processors, Computers and Industrial Engineering 34 (1998) 531–546.
[11] S. Hasija, C. Rajendran, Scheduling in flowshops to minimize total tardiness of jobs, International Journal of Production Research 42 (2004) 2289–2301.
[12] M. Tandon, P.T. Cummings, M.D. LeVan, Flowshop sequencing with non-permutation schedules, Computers and Chemical Engineering 15 (1991) 601–607.
[13] C. Koulamas, A new constructive heuristic for the flowshop scheduling problem, European Journal of Operational Research 105 (1998) 66–71.
[14] S. Pugazhendhi, S. Thiagarajan, C. Rajendran, N. Anantharaman, Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems, Computers and Industrial Engineering 44 (2003) 133–157.
[15] S.W. Lin, K.C. Ying, Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems, International Journal of Production Research 46 (2008) 1–14.
[16] R. Swaminathan, M.E. Pfund, J.W. Fowler, S.J. Mason, A.R. Keha, Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times, Computers and Operations Research 34 (2007) 3055–3068.
[17] C.J. Liao, L.M. Liao, C.T. Tseng, A performance evaluation of permutation vs. non-permutation schedules in a flowshop, International Journal of Production Research 44 (2006) 4297–4309.
[18] E.F. Stafford Jr., F.T. Tseng, On 'Redundant' constraints in Stafford's MILP model for the flowshop problem, Journal of the Operational Research Society 54 (2003) 1102–1105.
[19] F.T. Tseng, E.F. Stafford Jr., J.N.D. Gupta, An empirical analysis of integer programming formulations for the permutation flowshop, Omega 32 (2004) 285–293.
[20] E.F. Stafford Jr., F.T. Tseng, J.N. D Gupta, Comparative evaluation of MILP flowshop models, Journal of the Operational Research Society 56 (2005) 88–101.
[21] H.M. Wagner, An integer linear-programming model for machine scheduling, Naval Research Logistics Quarterly 6 (1959) 131–140.
[22] J.M. Wilson, Alternative formulations of a flowshop scheduling problem, Journal of the Operational Research Society 40 (1989) 395–399.
[23] A.S. Manne, On the job-shop scheduling problem, Operations Research 8 (1960) 219–223.
[24] C.J. Liao, C.T. You, An improved formulation for the job shop scheduling problem, Journal of the Operational Research Society 43 (1992) 1047–1054.
[25] W.B. Carlton, J.W. Barnes, A note on hashing functions and tabu search algorithms, European Journal of Operational Research 95 (1996) 237–239.
[26] F. Glover, E. Taillard, D. deWerra, A user's guide to tabu search, Annals of Operations Research 41 (1993) 3–28.