J. Opl Res. Soc. Vol. 44, No. 1, pp. 19-28
Printed in Great Britain.

0160-5682/93 $8.00+0.00

Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness

YEONG-DAE KIM
Department of Industrial Engineering, Korea Advanced Institute of Science and Technology

Several heuristics are presented for the flowshop scheduling problem with the objective of minimizing mean tardiness. We consider the cases in which job sequences on all machines are the same (permutation flowshop) and in which they may be different. For the former case, the various methods that have been devised for minimizing the makespan are modified for our objective, while the list scheduling algorithm is used for the latter case. These heuristics are tested and compared with each other on randomly-generated test problems.

Key words: heuristics, scheduling, sequencing

INTRODUCTION

A flowshop consists of m machines and n independent jobs that must be processed on the machines in the same order. In other words, a job consists of m operations and the jth operation of all jobs must be processed on machine j. The scheduling problem is to specify the order and the timing of the processing of the jobs on the machines, with an objective or objectives, assuming: a machine processes only one job at a time; a job can be processed on only one machine at a time; the operations are not pre-emptable; and set-up times of the operations are independent of the sequences and therefore they can be included in the processing time.

There are $(n!)^m$ different alternatives for ordering jobs on machines. In most research, however, only a subset of these alternatives is considered under the assumption that the operating sequences of the jobs are the same on every machine. In this case, the number of alternatives reduces to $n!$, and the schedules that satisfy this assumption are called *permutation schedules*. Although permutation schedules do not always include an optimal schedule, the importance of permutation schedules cannot be underestimated. This is because only permutation schedules are feasible in many real situations and it is easier to devise a method to find a good permutation schedule, than a method to find a good schedule which is not a permutation schedule.

Since it is unlikely that an efficient method can be found for solving the flowshop problem optimally[1], various heuristic methods have been developed for the problem (those by Ashour[2], Barany[3], Bonney and Gundry[4], Campbell *et al.*[5], Dannenbring[6], Gupta[7,8], Ho and Chang[9], Hundal and Rajgopal[10], King and Spachis[11], Nawaz *et al.*[12], Ogbu and Smith[13], Osman and Potts[14], Page[15], Palmer[16], Park *et al.*[17], Taillard[18], and Widmer and Hertz[19], for example). The performance criterion used in these articles is the maximum completion time (makespan).

There are a few papers that deal with objectives involving the due dates of jobs. Gelders and Sambandam[20] suggested four heuristics with the objective of minimizing the sum of weighted tardiness and flow time, and Ow[21] developed a scheduling procedure to minimize the sum of weighted tardiness for a special case in which the processing times are proportionate. On the other hand, Grabowski *et al.*[22], Sen *et al.*[23], and Townsend[24] sought optimal solutions with branch-and-bound techniques. Sen *et al.*[23] used, as an objective, minimizing the mean tardiness on two-machine flowshops, while Grabowski *et al.*[22] used the maximum lateness among all jobs as the performance criterion, and Townsend[24] tried to minimize the maximum tardiness on m-machine flowshops.

In this paper, we devise heuristic algorithms for the flowshop problem with the objective of minimizing the mean tardiness. There may be two approaches to developing algorithms for

Correspondence: Y-D. Kim, Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, 371-1 Gusung-Dong, Yusong-Gu, Daejon 305-701, Korea

flowshop scheduling problems: one in which the solutions are not restricted to be permutation schedules; and one in which only permutation schedules are considered. Since the permutation schedules are not dominant in the m-machine flowshop problem that has the objective of minimizing mean tardiness when $m \geqslant 3$, the former approach can give better schedules. However, it is difficult to develop an algorithm with the former approach, not only because many more alternatives have to be considered but also because there are no known properties of optimal schedules.

There have been many methods devised with the second approach, although the objective is usually that of minimizing the makespan. Some of these methods can be modified for the objectives considering the due dates of jobs. On the other hand, methods that use the former approach are very rare even for the objective of minimizing the makespan. Since the flowshop scheduling problem is a special case of the jobshop problem, we can apply jobshop scheduling methods to flowshops. Due dates are considered in many existing jobshop scheduling methods.

In general, the performance of heuristics can be analysed in several ways, such as: experimental analysis with tests on sets of random problems; worst-case analysis; and probabilistic analysis. For the flowshop problems, the first type of analysis is used in most research; while Barany[3], Gonzalez and Sahni[25], and Nowicki and Smutnicki[26] perform the second type of analysis. There seems to be no known results of the latter type of analysis for flowshop problems. In this paper, we evaluate suggested heuristics with an experimental analysis on randomly-generated test problems.

HEURISTICS CONSIDERING ALL SCHEDULES

Due to the difficulty of devising special heuristics of this type, we apply heuristic jobshop scheduling methods to our flowshop problems. *List scheduling* is a class of heuristic algorithms commonly used in multimachine scheduling problems, especially in jobshop scheduling. Such an algorithm lists the operations that are available for each machine and then assigns the operations in some order to the machines as the machines become available. The order of the assignment can be generated randomly or from priority rules. In list scheduling for static flowshops, all jobs can be scheduled on machine k without considering machines $k+1, k+2, \ldots, m$. Note that this is impossible for jobshops since the sequences of machines that the job visit may vary.

Various priority rules have been devised and tested for jobshops with or without consideration of due dates[27–35]. Blackstone *et al.*[36] and Panwalker and Iskander[37] gave extensive surveys of the rules. These rules can be directly applied to our flowshop problems even though they do not take advantage of the special characteristics of flowshops.

Here, we test seven of the rules that are simple or known to work better than others. The priority rules tested here are SPT (shortest processing time), MDD (modified due date), MOD (modified operation due date), SLACK, S/RMWK (slack per remaining work), COVERT (cost over time), and ATC (apparent tardiness cost) rules. In these rules, when a machine (say machine j) becomes free, an available job, which has the minimum value of the corresponding priority measure, will be assigned to the machine. The priority measure of (jth operation of) job i awaiting machine j is defined below. Here, p_{ij} denotes the processing time of the jth operation of job i, d_i is due time of job i, t is the time at which the priority rules are applied or an operation is to be selected for machine j, and r_i is the remaining work of job i, i.e. the sum of processing times of the jth through mth operations. Also, let x^+ denote $\max(0, x)$.

$$\text{SPT: } p_{ij}$$

$$\text{SLACK: } d_i - r_i - t$$

$$\text{S/RMWK: } (d_i - r_i - t)/r_i$$

$$\text{MDD: } \max\{d_i, t + r_i\}$$

$$\text{MOD: } \max\{d_i - b(r_i - p_{ij}), t + r_i\}$$

$$\text{COVERT: } -\left[\left\{1 - \frac{(d_i - r_i - t)^+}{kbr_i}\right\}\Big/ p_{ij}\right]^+$$

ATC: $-\exp[-\{d_i - b(r_i - p_{ij}) - p_{ij} - t\}^+/k\bar{p}]/p_{ij}$

where $\bar{p}$ is the average processing time of the waiting operations.

In the last three rules, b and k are parameters that must be specified. These parameters are used to take into account the waiting time of operations in a queue, and the machine utilization when estimating the completion time of a job. This is because the sum of the processing times of an operation and its successors is not an accurate estimate of the completion time of the corresponding job. The parameter b can be called the leadtime estimation parameter, as it takes into account the waiting time between operations; while k is called the adjustment multiplier; it adjusts the expected waiting time to the worst case, for example the 95th percentile of the cumulative probability distribution. See Russell *et al.*[33] and Vepsalainen and Morton[35] for more detail of these parameters.

When generating a schedule using one of these list scheduling algorithms, we may have to consider an important property of flowshop scheduling problems using a regular measure: with respect to any regular measure of performance, there exists an optimal schedule in which the same job sequence occurs on machines 1 and 2. See Baker[38] for a definition of the regular measure.

Since our measure (mean tardiness) is regular, we only have to consider schedules in which the same job sequence occurs on machines 1 and 2. One way of generating such schedules in list scheduling algorithms is to apply the FIFO (first-in first-out) rule when selecting jobs on machine 2 regardless of the priority rules used in the algorithms. However, this method may not always give better solutions than the method in which the priority rules of the algorithms are used on all machines, since the sequence on machine 1 may not be a good one for machine 2.

Another way is to rearrange the jobs on the first machine according to the sequence on the second machine after completing the list scheduling algorithms. Notice that for any schedule in which the sequences of jobs on machines 1 and 2 are different, we can always improve, or at least do not deteriorate the original solutions with this method (this can be seen from the proof for the above property[38]). However, this rearrangement does not improve list scheduling algorithms in which machines are not kept idle when there are jobs waiting for them. The completion times of jobs on machine 2 remain the same even after the rearrangement of the jobs on machine 1. Consequently, the value of any regular measure of a schedule does not change with this method. We will compare these two methods in our computational experiments.

HEURISTICS CONSIDERING ONLY PERMUTATION SCHEDULES

Various heuristics have been developed for permutation flowshops, with the objective of minimizing the makespan. These heuristics can be classified into several categories. These include: heuristics using special properties of the problem, such as by defining priorities of jobs[7,10,15,16,22], by transforming to simpler or relaxed problems such as two-machine problems[5,6,17,25], and by other methods[2,4,8,9,11,17]; and heuristics that directly seek a good sequence, such as by constructing sequences through job insertions[12,18] and by improving a given sequence through neighbourhood search methods. Some of the neighbourhood search methods used in previous research include simple search methods[6,39], taboo search methods[18,19], and simulated annealing methods[13,14].

While the heuristics in the first category are tailored to a specific objective (minimizing makespan), those in the second category can be directly used for any regular performance measures. Also, it is known through the computational experiments of many researchers, that heuristics in the second category give better solutions than those in the first category. The only drawback is that they need more computation time. However, this is negligible compared with the time needed for an optimal solution. In this research, we adapt heuristics in the second category to the problem of minimizing mean tardiness.

We test here five heuristics that have been modified from previously-suggested algorithms such as those of Gelders and Sambandam[20], Nawaz *et al.*[12], and Widmer and Hertz[19]. The five heuristics are GS2, NEHedd, NEHldd, ENS (extensive neighbourhood search), and TS (taboo search). We briefly describe each of the algorithms, in the following.

(1) *GS2* (*The second heuristic of Gelders and Sambandam*)
Gelders and Sambandam give four heuristics for the performance measure of the sum of weighted tardiness and weighted flowtime. This algorithm uses a job dispatching rule to obtain a sequence. Higher priorities are given to jobs that are more expensive to hold (to schedule later). They suggest four methods of computing the priorities of jobs. Since the second method among them is argued to be the best for the mean tardiness problem, it is selected for our test.

(2) *NEHedd* (*Algorithm of Nawaz, Enscore and Ham with EDD rule for preprocessing*)
Nawaz *et al.*[12] present a simple but very good algorithm for minimizing makespan (called the NEH algorithm by other researchers). In the original NEH algorithm, jobs are sorted in decreasing (non-increasing) order of the sum of the processing times on machines, then a final solution is obtained in a constructive way, adding at each step a new job in that order, and then inserting it in the best place, i.e. the one that results in the best partial solution. Since due dates are considered in our problem, there may be various methods for sorting jobs. In NEHedd, jobs are sorted in non-decreasing order of due dates. EDD (edd) denotes earliest due date.

(3) *NEHldd* (*Algorithm of Nawaz, Enscore and Ham with LDD rule for preprocessing*)
In this algorithm, jobs are sorted in non-increasing order of due dates (latest due date rule). Since the jobs with later due dates are considered earlier, if the scheduled jobs in a partial solution are assumed to be placed at the beginning of the whole sequence, many or all alternatives result in zero tardiness. To prevent this, we have to assume the scheduled jobs are to be placed at the end of the whole sequence and then compute the tardiness of the jobs for the alternatives, with a restriction on the time at which each of the machines is available for those jobs. These machine-available times can be estimated with lower bounds on the completion times of the machines, for a given set of jobs such as those given by Baker[38]. The bound used in this algorithm is $\max_{1 \leqslant j \leqslant k} \left\{ \sum_{i \notin \sigma} p_{ij} + \min_{i \notin \sigma} \sum_{l=1}^{j-1} p_{il} + \min_{i \notin \sigma} \sum_{l=j+1}^{k} p_{il} \right\}$ for machine k, where σ is the set of jobs in the partial solution, including the job being added.

(4) *ENS* (*Extensive neighbourhood search method*)
This algorithm starts from an initial full sequence (for example, the EDD sequence) and tries to improve the solution by moving to a best neighbouring solution. A neighbouring solution of a given solution is obtained by interchanging a pair of jobs in the given solution. Therefore, we have $n(n-1)/2$ neighbouring solutions for each solution, where n is the number of jobs. The best neighbouring solution is the solution with the minimum total tardiness. The algorithm terminates when the best neighbouring solution is no better than the given solution.

(5) *TS* (*Taboo search method of Widmer and Hertz*)
Differently from ENS, a move can be made to the best neighbouring solution even if it is worse than the given solution. In this method, however, a list L of taboo moves—the moves that are not allowed at the present iteration—is kept and updated when a move is made. For example, if job x is in the ith place and y is in the jth place before a move and they are exchanged after the move, the pairs (x, i) and (y, j) are introduced in L so that jobs x and y may not return to the positions i and j, respectively for a certain number of iterations. When a pair of taboo moves are included in L, the two oldest pairs of taboo moves in L are removed. The algorithm terminates when no improvements have been made for a certain number of iterations (say $I_{\max}$). Note that the size of the list of taboo moves ($|L|$) and $I_{\max}$ affects the solution quality and computation time for the algorithm. We set, in the test, $|L| = 7$ and $I_{\max} = \min(n, 15)$, where n is the number of jobs.

See the corresponding references for more details of GS2, NEH, and the taboo search methods. Note that other preprocessing methods can be used for a NEH method such as SPT (shortest processing time) and LPT (longest processing time), in which jobs are sorted in non-decreasing and non-increasing order of the sum of the processing times on all machines, respectively. (The original NEH method for minimizing the makespan uses the LPT rule for preprocessing).

The list scheduling methods can also be modified for the permutation flowshop problems. In this case, the priorities of the jobs are determined at the time when machine 1 becomes available. Priority rules such as EDD (earliest due date), SLACK, S/RMWK, and MDD rules can be used whilst maintaining the basic ideas of the rules.

The list scheduling algorithms with these priority rules include one job at a time into the set of jobs scheduled so far (initially empty) until all jobs are included (or scheduled). At each time when machine 1 becomes available, these algorithms estimate the completion time of each of the unscheduled jobs (the jobs that are not yet included in the set) if each job is scheduled next. The exact completion time can be obtained using the available time of each of the machines, i.e. the completion time of all scheduled jobs on each of the machines, which can be computed easily since the sequence of the scheduled jobs is known. Then, the slack time of each job can be estimated by the difference of the due date and the completion time of the job. With this slack time, we obtain the priorities of the unscheduled jobs.

Let σ be the sequence of jobs that are scheduled so far and t be the time at which jobs are considered for selection, i.e. $t = \sum_{i \in \sigma} p_{i1}$. Also, let $C_i(\sigma)$ be the completion time of job $i \notin \sigma$ if it were scheduled just after the sequence σ. Then, at time t, the job with the minimum value of the following is selected:

$$\text{EDDp: } d_i;$$

$$\text{SLACKp: } d_i - C_i(\sigma);$$

$$\text{S/RMWKp: } (d_i - C_i(\sigma)) \Big/ \sum_{j=1}^{m} p_{ij};$$

$$\text{MDDp: } \max\{d_i, C_i(\sigma)\}.$$

Note that algorithm GS2 is also a list scheduling method, even though it was presented earlier in this section. It uses a priority measure that is more sophisticated than the above four rules.

Solutions from GS2, NEHedd, NEHldd, and the above four list scheduling methods can be improved by pairwise interchanges between adjacent jobs (the solutions from ENS and TS cannot be improved by this pairwise interchange).

COMPUTATIONAL EXPERIMENTS

Before comparing the heuristics presented earlier, some preliminary tests are made on a number (200) of test problems. The results of these tests are as follows.

(1) The methods of generating schedules in which the same job sequence occurs on machines 1 and 2 had no significant effect on the solution quality, although the second method (to rearrange the jobs on the first machine according to the sequence on the second machine after completing the list scheduling algorithms) gave better solutions on slightly more problems than the first method (to apply the FIFO rule when selecting jobs on machine 2 regardless of the priority rules).

(2) MOD with $b = 1.25$ gave better results than MOD with other values for this parameter. COVERT performed best when $k = 1.0$, $b = 1.0$ while $k = 2.0$, $b = 1.0$ was the best combination for ATC.

(3) NEH methods with SPT and LPT rules for preprocessing were, by far, outperformed by NEH with EDD and LDD rules.

To test the algorithms under their best conditions, as identified in the above results, 1000 problem instances were generated randomly. In the problems, the number of jobs ranges from 15 to 50 and the number of machines ranges from 5 to 25. Processing times and due dates are generated following the method by Potts and Van Wassenhove[40], which is often used for single-machine scheduling problems considering due dates. The processing time of an operation is generated from a discrete uniform distribution with a range 1 to 35. Then, due dates of the jobs are generated using two parameters, R (relative range of due dates) and T (tardiness factor). The due dates are generated from a discrete uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$, where P is the time at which all jobs are completed. This time is estimated with a lower bound on the makespan[38] that can be obtained from the processing times of the operations. T ranges from 0.1 to 0.5 and R ranges from 0.8 to 1.8 in the problems.

Since optimal solutions cannot be obtained in a reasonable amount of time, we compare the

solutions with the *relative deviation index*, which is defined as $(T_a - T_B)/(T_W - T_B)$ for algorithm a, where T_a, T_B, and T_W are the solution values of algorithm a, of the algorithm which gave the smallest solution value, and of the algorithm which gave the largest solution value, respectively. Therefore, the index has a value between 0 and 1. This measure is used instead of the commonly-used performance ratio, i.e. T_a/T_o, where T_o is the optimal solution value. This is because optimal solutions cannot be obtained easily, and if the best solution value is a small number (which can occur in tardiness problems), the performance ratio may underestimate the performance of an algorithm which has a solution that is only slightly worse than the best one.

All the heuristics are coded in FORTRAN and run on a personal computer with an 80387 processor. Table 1 shows the performance of the 16 algorithms (including, for seven of them, before and after pairwise interchanges between adjacent jobs) with two performance measures: relative deviation index and number of problems for which each algorithm found the best solution. In the table, LS and LSP denote the set of the seven list scheduling algorithms considering all schedules and the set of the four list scheduling algorithms generating permutation schedules, respectively. The 'best' results in the table are computed from the best solution among the solutions of the list scheduling algorithms.

TABLE 1. *Performance of the algorithms*

		Before adjacent pairwise interchange			After adjacent pairwise interchange		
		Number of best solutions	Relative deviation index		Number of best solutions	Relative deviation index	
	Algorithms		Mean	Standard deviation		Mean	Standard deviation
LS	SPT	0	0.888	0.188			
	MDD	108	0.257	0.215			
	MOD	98	0.310	0.247			
	SLACK	91	0.491	0.382			
	S/RMWK	80	0.445	0.337			
	COVERT	49	0.307	0.240			
	ACT	105	0.291	0.243			
	best	118	0.209	0.177			
LSP	EDDp	105	0.438	0.347	154	0.233	0.221
	SLACKp	101	0.470	0.369	147	0.233	0.222
	S/RMWKp	75	0.357	0.277	144	0.205	0.192
	MDDp	107	0.290	0.255	152	0.212	0.204
	best	118	0.265	0.230	190	0.153	0.155
	GS2	16	0.367	0.214	51	0.284	0.207
	NEHedd	200	0.046	0.056	363	0.034	0.048
	NEHldd	144	0.251	0.300	189	0.196	0.235
	ENS	323	0.033	0.048			
	TS	737	0.007	0.020			

Among the algorithms in LS, MDD showed the best performance. This result is different from that of studies on jobshops, in which ATC or COVERT are known to work better than others. Among the algorithms for permutation schedules, MDDp slightly outperformed the other three algorithms. Moreover, it gave much better results than GS2, a list scheduling algorithm using very sophisticated priority rules. (This shows that the performance of an algorithm is not directly related to its complexity.) A comparison of the best solutions from the two sets of list scheduling algorithms (the best of LS slightly outperformed the best of LSP) reaffirms the fact that the permutation schedules are not dominant. However, after pairwise interchanges between adjacent jobs, LSP worked better than LS. The ease by which schedules can be improved by job interchanges is a major advantage of the algorithms in LSP over those in LS.

The NEH method with the EDD rule for preprocessing was better than the one with the LDD rule. Moreover, it worked better than all list scheduling algorithms including GS2, and after pairwise interchanges it performed almost the same as ENS. All of these algorithms were outperformed by the taboo search method. However, for some problems (about a quarter of the test problems), the solutions from TS (using the EDD sequence as the initial sequence) were worse than the other algorithms, such as NEHedd.

TABLE 2. *CPU time for the algorithms*

Number of jobs	Number of machines	LS	LSP	GS2	NEHedd	NEHldd	ENS	TS
15	5	0.11	0.03(0.10)	0.19(0.03)	0.07(0.01)	0.08(0.01)	0.41	1.99
15	10	0.17	0.04(0.20)	0.50(0.04)	0.13(0.02)	0.15(0.02)	1.07	4.88
15	15	0.24	0.05(0.31)	0.93(0.05)	0.18(0.03)	0.22(0.03)	1.63	6.65
15	20	0.30	0.06(0.41)	1.48(0.07)	0.23(0.05)	0.28(0.05)	2.51	8.82
15	25	0.36	0.07(0.51)	2.14(0.09)	0.29(0.06)	0.35(0.12)	3.34	11.52
20	5	0.18	0.04(0.18)	0.44(0.05)	0.16(0.03)	0.17(0.02)	1.24	5.43
20	10	0.29	0.06(0.37)	1.16(0.07)	0.28(0.05)	0.32(0.05)	3.24	11.53
20	15	0.38	0.09(0.54)	2.16(0.10)	0.41(0.07)	0.46(0.07)	5.13	17.52
20	20	0.49	0.10(0.75)	3.45(0.13)	0.52(0.08)	0.62(0.09)	7.39	25.17
20	25	0.60	0.12(0.89)	5.00(0.16)	0.65(0.11)	0.77(0.23)	9.53	34.10
30	5	0.36	0.09(0.38)	1.44(0.09)	0.53(0.05)	0.54(0.06)	4.13	16.63
30	10	0.59	0.12(0.86)	3.82(0.16)	0.92(0.11)	0.98(0.12)	13.01	44.56
30	15	0.80	0.18(1.33)	7.14(0.27)	1.33(0.16)	1.43(0.17)	22.68	77.02
30	20	1.01	0.22(1.84)	11.45(0.32)	1.73(0.21)	1.88(0.23)	34.98	96.54
30	25	1.21	0.27(2.26)	16.64(0.43)	2.13(0.24)	2.32(0.59)	44.38	128.09
40	5	0.62	0.15(0.68)	3.35(0.16)	1.17(0.10)	1.22(0.11)	11.12	38.45
40	10	1.00	0.12(1.49)	8.96(0.32)	2.10(0.21)	2.20(0.21)	34.60	98.91
40	15	1.35	0.21(2.41)	16.76(0.47)	3.04(0.27)	3.20(0.30)	60.32	186.55
40	20	1.70	0.40(3.32)	26.89(0.59)	3.98(0.39)	4.20(0.42)	99.21	251.24
40	25	2.06	0.48(4.02)	38.07(0.77)	4.92(0.43)	5.22(1.15)	133.01	361.55
50	5	0.96	0.23(1.03)	6.13(0.27)	2.27(0.14)	2.33(0.15)	23.32	74.31
50	10	1.52	0.35(2.36)	16.32(0.57)	4.06(0.31)	4.20(0.32)	84.88	219.87
50	15	2.05	0.49(3.72)	30.44(0.84)	5.86(0.50)	6.08(0.51)	143.53	360.02
50	20	2.57	0.61(5.17)	50.99(1.14)	7.69(0.62)	8.00(0.65)	213.69	498.95
50	25	3.12	0.74(6.75)	76.08(1.25)	9.51(0.74)	9.94(2.24)	281.87	705.76

The figures in parentheses denote the CPU times for adjacent pairwise interchanges.

Table 2 shows the CPU time needed for each algorithm for different problem sizes. The CPU times for the list scheduling algorithms are shown together and denoted by LS and LSP, since those for the individual algorithms are too short to be timed by the clock of the personal computer. Generally speaking, there were positive correlations between the CPU times for the algorithms and their performance, except for GS2, which worked poorly although it needed a long computation time.

As expected, the two sets of list scheduling algorithms required very short computation times. On the contrary, GS2, which is also a list scheduling method, needed considerably longer time because of its sophisticated method of calculating the priorities of jobs. The CPU times for the NEH methods were much shorter than those for GS2 and the two search methods, ENS and TS. The taboo search method, which requires a very long time for a problem of medium size, may not be appropriate for circumstances in which a quick decision is needed, such as real-time scheduling situations. The time for adjacent pairwise interchanges was short compared with the time for the main procedures of the algorithms, except for the list scheduling algorithms. This is because the solutions from the list scheduling algorithms have more room for improvement than other algorithms.

To study the effect of tightness of due dates on the performance of the algorithms, we prepared Tables 3 and 4. In Table 3, the (relative) performance of the algorithms is given for different tardiness factors. Although TS dominated the other algorithms for all cases, it was less dominant when the tardiness factors were smaller (due dates are not tight). This is because when due dates are not tight, a small alteration of a sequence does not change the total tardiness and therefore TS terminates in many cases after fruitless attempts for improvements. Similar results can be seen from Table 4. When the due-date range is small (in other words due dates are tighter), TS is more dominant. In cases in which due dates are tight, a small change in a sequence can often reduce the total tardiness, and therefore TS may improve the original sequences over and over again.

If one of these heuristic scheduling methods has to be selected for use in practice, we have to consider the performance (quality of schedules) and the speed of the algorithms. NEHedd with adjacent pairwise interchanges might be a good candidate for many flowshops.

TABLE 3. *Performance of the algorithms for different tatdiness factors*

	T Number of problems tested				
Algorithms	0.1 300	0.2 250	0.3 200	0.4 150	0.5 100
LS best	0.14/82	0.17/28	0.23/7	0.29/1	0.35/0
LSP best	0.17/81 (0.09/116)	0.22/29 (0.12/53)	0.29/7 (0.17/18)	0.37/1 (0.23/3)	0.46/0 (0.29/0)
GS2	0.29/16 (0.21/40)	0.36/0 (0.27/8)	0.41/0 (0.33/1)	0.43/0 (0.35/0)	0.43/0 (0.36/2)
NEHedd	0.03/109 (0.02/150)	0.04/51 (0.03/96)	0.05/26 (0.04/67)	0.06/8 (0.05/31)	0.06/6 (0.05/19)
NEHldd	0.22/88 (0.17/102)	0.23/43 (0.18/60)	0.26/11 (0.20/19)	0.29/1 (0.23/5)	0.31/1 (0.26/3)
ENS	0.02/152	0.03/92	0.04/46	0.04/20	0.05/13
TS	0.01/228	0.01/176	0.01/141	0.01/115	0.01/77

1. At each entry of the table *r/n*, *r* denotes the mean relative deviation index and *n* denotes the number of best solutions each algorithm found.
2. The figures in parentheses are results from the solutions after adjacent pairwise interchange.

TABLE 4. *Performance of the algorithms for different due-date ranges*

	R Number of problems tested					
Algorithms	0.8 250	1.0 250	1.2 200	1.4 150	1.6 100	1.8 50
LS best	0.27/27	0.24/29	0.19/23	0.17/23	0.13/10	0.11/6
LSP best	0.33/28 (0.20/36)	0.30/29 (0.18/43)	0.24/25 (0.14/42)	0.21/23 (0.12/38)	0.18/6 (0.10/19)	0.16/7 (0.08/12)
GS2	0.43/5 (0.35/13)	0.39/2 (0.30/11)	0.35/5 (0.27/10)	0.32/4 (0.23/12)	0.31/0 (0.22/2)	0.25/0 (0.17/3)
NEHedd	0.06/34 (0.05/73)	0.05/42 (0.04/73)	0.04/46 (0.03/81)	0.03/40 (0.02/66)	0.03/24 (0.03/45)	0.02/14 (0.02/25)
NEHldd	0.27/32 (0.22/39)	0.27/33 (0.21/43)	0.24/30 (0.18/38)	0.23/25 (0.18/35)	0.22/14 (0.16/19)	0.24/10 (0.19/15)
ENS	0.04/64	0.04/73	0.04/66	0.03/57	0.02/40	0.02/23
TS	0.01/198	0.01/198	0.01/146	0.01/99	0.01/62	0.01/34

See the footnotes of Table 3.

CONCLUDING REMARKS

We have considered flowshop scheduling problems with the objective of minimizing mean tardiness. Two approaches for developing heuristics were discussed. The list scheduling methods can be used to generate a good schedule that is not restricted to be a permutation schedule. To find a good permutation schedule, we can use many heuristics developed for minimizing the makespan. Also, list scheduling methods can be adapted for this approach.

Various heuristics devised from these two approaches were compared on randomly-generated test problems. In the tests, schedules that are not permutation schedules were slightly better than permutation schedules from the list scheduling methods, since permutation schedules consider only a subset of alternatives for job sequencing. However, this deficiency was overcome by a typical advantage of permutation schedules—the ease of improvement. (Research on methods for improving solutions is needed for the approach of generating schedules that are not permutation schedules.) Also, the results show that the neighbourhood search algorithms such as taboo search outperformed the others at the expense of a computational burden.

In this research, a class of neighbourhood search methods—the simulated annealing method—is left out from the computational comparisons. As can be seen in Osman and Potts[14], this method needs various parameters (such as the *temperature* and the number of iterations) or methods (of

constructing a neighbourhood and of searching a neighbourhood) to be carefully selected to give the best performance for each problem to which it is applied. Therefore, applying this method to our problem would be another independent research topic. Also, an additional study might be needed on how to construct a neighbouring solution and to select an appropriate search method for ENS and TS. In addition, what type of neighbourhood is preferable for a local search after individual heuristics, may be a question that still needs to be answered.

In this paper, we presented several algorithms and compared their performance, i.e. the solution quality and computation time. These test results are expected to be useful when choosing an algorithm that is appropriate for the system's demands.

Acknowledgement—The author wuold like to thank one referee for valuable suggestions that have improved the content and exposition of the paper. Many of the referenced papers are due to that referee.

REFERENCES

1. S. French (1982) *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Halsted Press, New York.
2. S. Ashour (1970) An experimental investigation and comparative evaluation of flow-shop scheduling techniques. *Opns Res.* **18**, 541–549.
3. I. Barany (1981) A vector-sum theorem and its application to improving flow shop guarantees. *Maths Opns Res.* **6**, 445–452.
4. M. C. Bonney and S. W. Gundry (1976) Solutions to the constrained flowshop sequencing problem. *Opl Res. Q.* **27**, 869–883.
5. H. G. Campbell, R. A. Dudek and M. L. Smith (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Mgmt Sci.* **16**, 630–637.
6. D. G. Dannenbring (1977) An evaluation of flow shop sequencing heuristics. *Mgmt Sci.* **23**, 1174–1182.
7. J. N. D. Gupta (1971) A functional heuristic algorithm for the flowshop scheduling problem. *Opl Res. Q.* **22**, 39–47.
8. J. N. D. Gupta (1972) Heuristic algorithms for multistage flowshop scheduling problem. *AIIE Trans.* **4**, 11–18.
9. J. C. Ho and Y-L. Chang (1991) A new heuristic for the n-job M-machine flow-shop problem. *Eur. J. Opl Res.* **52**, 194–202.
10. T. S. Hundal and J. Rajgopal (1988) An extension of Palmer's heuristic for the flow shop scheduling problem. *Int. J. Prod. Res.* **26**, 1119–1124.
11. J. R. King and A. S. Spachis (1980) Heuristics for flow-shop scheduling. *Int. J. Prod. Res.* **18**, 345–357.
12. M. Nawaz, E. E. Enscore Jr. and I. Ham (1983) A heuristic algorithm for the m-machine, n-job flow-shop scheduling problem. *Omega* **11**, 91–95.
13. F. A. Ogbu and D. K. Smith (1990) The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. *Comps and Opns Res.* **17**, 243–253.
14. I. H. Osman and C. N. Potts (1989) Simulated annealing for permutation flow-shop scheduling. *Omega* **17**, 551–557.
15. E. S. Page (1961) An approach to the scheduling of jobs on machines. *J. Roy. Stat. Soc.* **23**, 484–492.
16. D. S. Palmer (1965) Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Opl Res. Q.* **16**, 101–107.
17. Y. B. Park, C. D. Pegden and E. E. Enscore (1984) A survey and evaluation of static flowshop scheduling heuristics. *Int. J. Prod. Res.* **22**, 127–141.
18. E. Taillard (1990) Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Opl Res.* **47**, 65–74.
19. M. Widmer and A. Hertz (1989) A new heuristic method for the flow shop sequencing problem. *Eur. J. Opl Res.* **41**, 186–193.
20. L. F. Gelders and N. Sambandam (1978) Four simple heuristics for scheduling a flow-shop. *Int. J. Prod. Res.* **16**, 221–231.
21. P. S. Ow (1985) Focused scheduling in proportionate flowshops. *Mgmt Sci.* **31**, 852–869.
22. J. Grabowski, E. Skubalska and C. Smutnicki (1983) On flow shop scheduling with release and due dates to minimize maximum lateness. *J. Opl Res. Soc.* **34**, 615–620.
23. T. Sen, P. Dileepan and J. N. D. Gupta (1989) The two-machine flowshop scheduling problem with total tardiness. *Comps and Opns Res.* **16**, 333–340.
24. W. Townsend (1977) Sequencing n jobs on m machines to minimise maximum tardiness: a branch-and-bound solution. *Mgmt Sci.* **23**, 1016–1019.
25. T. Gonzalez and S. Sahni (1978) Flowshop and jobshop schedules: complexity and approximation. *Opns Res.* **26**, 36–52.
26. E. Nowicki and C. Smutnicki (1989) Worst-case analysis of an approximation algorithm for flow-shop scheduling. *Opns Res. Lett.* **8**, 171–177.
27. K. R. Baker and J. J. Kanet (1983) Job shop scheduling with modified due dates. *J. Opns Mgmt* **4**, 11–22.
28. J. B. Berry and V. Rao (1975) Critical ratio scheduling: an experimental analysis. *Mgmt Sci.* **22**, 192–201.
29. E. M. Dar-El and R. A. Wysk (1982) Job shop scheduling—a systematic approach. *J. Manuf. Syst.* **1**, 77–88.
30. D. A. Elvers and L. R. Taube (1983) Time completion for various dispatching rules in job shops. *Omega* **11**, 81–89.
31. Y-D. Kim (1987) On the superiority of a backward approach in list scheduling algorithms for multi-machine makespan problems. *Int. J. Prod. Res.* **25**, 1751–1759.

32. P. J. O'GRADY and C. HARRISON (1985) A general search sequencing rule for job shop sequencing. *Int. J. Prod. Res.* **23**, 961–973.
33. R. S. RUSSELL, E. M. DAR-EL and B. S. TAYLOR III (1987) A comparative analysis of the COVERT job sequencing rule using various shop performance measures. *Int. J. Prod. Res.* **25**, 1523–1540.
34. G. D. SCUDDER and T. R. HOFFMANN (1985) Composite cost-based rules for priority scheduling in a randomly routed job shop. *Int. J. Prod. Res.* **23**, 1185–1195.
35. A. P. J. VEPSALAINEN and T. E. MORTON (1987) Priority rules for job shops with weighted tardiness costs. *Mgmt Sci.* **33**, 1035–1047.
36. J. H. BLACKSTONE Jr., D. T. PHILLIPS and G. L. HOGG (1982) A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *Int. J. Prod. Res.* **20**, 27–45.
37. S. S. PANWALKAR and W. ISKANDER (1977) A survey of scheduling rules. *Opns Res.* **25**, 45–61.
38. K. BAKER (1974) *Introduction to Sequencing and Scheduling.* Wiley, New York.
39. M. J. KRONE and K. STEIGLITZ (1974) Heuristic-programming solution of a flowshop-scheduling problem. *Opns Res.* **22**, 629–638.
40. C. N. POTTS and L. N. VAN WASSENHOVE (1982) A decomposition algorithm for the single machine total tardiness problem. *Opns Res. Lett.* **1**, 177–181.