Int J Adv Manuf Technol (2008) 38:1018–1025
DOI 10.1007/s00170-007-1156-z

ORIGINAL ARTICLE

An efficient heuristic approach to total flowtime minimization in permutation flowshop scheduling

Dipak Laha · Uday K. Chakraborty

Received: 15 January 2007 / Accepted: 9 July 2007 / Published online: 14 August 2007

Abstract The problem of permutation flowshop scheduling is considered with the objective of minimizing the total flowtime. We present a constructive heuristic and two composite heuristics to solve the problem. The composite heuristics combine the simulated annealing method of Chakravarthy and Rajendran [Production Planning and Control 10 (1999)], the constructive heuristic of Nawaz et al. [Omega 11 (1983)] and the new heuristic. Computational analysis is carried out with the benchmark problems of Taillard [European Journal of Operational Research 64 (1993)]. The two composite heuristics produce better quality solutions than those produced by the composite heuristics of Liu and Reeves [European Journal of Operational Research 132 (2001)]. Statistical tests of significance are used to substantiate the improvement in solution quality.

Keywords Heuristics · Scheduling · Flowshop · Flowtime

1 Introduction

The problem of the assignment of times to a set of jobs for processing through a series of machines has long received the attention of researchers. A great deal of research has been carried out in manufacturing scheduling. The practical importance of such problems is great, as scheduling plays a significant role in successful production planning and control. A variety of scheduling algorithms have been developed over the past several years to address different production systems. Two common problems that frequently appear in the scheduling literature are flowshop scheduling and jobshop scheduling. In flowshop scheduling, it is generally assumed that the jobs must be processed on the machines in the same technological or machine order. In jobshop scheduling, however, jobs are commonly processed following different machine orders.

In the flowshop scheduling problem, n jobs are to be processed on m machines. The order of the machines is fixed. We assume that a machine processes one job at a time and a job is processed on one machine at a time without preemption. Let $t_p(i, j)$ denote the processing time of job j on machine i, and $t_c(i, j)$ denote the completion time of job j on machine i. Let J_j denote the j-th job and M_i the i-th machine. The completion times of the jobs are obtained as follows:

$$\text{For } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n$$
$$t_c(M_1, J_1) = t_p(M_1, J_1)$$
$$t_c(M_i, J_1) = t_c(M_{i-1}, J_1) + t_p(M_i, J_1)$$
$$t_c(M_1, J_j) = t_c(M_1, J_{j-1}) + t_p(M_1, J_j)$$
$$t_c(M_i, J_j) = \max\{t_c(M_{i-1}, J_j), t_c(M_i, J_{j-1})\} + t_p(M_i, J_j)$$

The *total flowtime is* defined as the sum of completion times of all the jobs in a schedule, $\sum_{j=1}^{n} t_c(M_m, J_j)$. The goal is to obtain the n-job sequence that minimizes the total flowtime of jobs. The control of the total flowtime is of great practical importance in today's production control departments, as a small total flowtime leads to stable or even utilization of resources, a rapid turn-around of jobs and minimization of work in process inventory [1, 2]. In

D. Laha (✉)
Mechanical Engineering Department, Jadavpur University,
Calcutta 700032, India
e-mail: dipaklaha_jume@yahoo.com

U. K. Chakraborty
Department of Mathematics and Computer Science,
University of Missouri,
St. Louis, MO 63121, USA
e-mail: uday@cs.umsl.edu

this context it should be mentioned that a second major line of research in flowshop scheduling involves designing improved heuristics (e.g., [3, 4, 5, 6]) for minimizing the *makespan,* which is defined as the completion time of the last job, $t_c(M_m, J_n)$.

For n jobs, the search space (for makespan minimization or total flow time minimization) consists of $n!$ possible job sequences ($n!$ permutations of n distinct objects). The problem of flowshop scheduling is NP-complete [7] and exhaustive enumeration of all $n!$ sequences is computationally prohibitive. Therefore, heuristic approaches rather than exact methods are the most suitable optimization methods to solve scheduling problems involving a large number of jobs.

The terms *sequence* (*schedule*) and *partial sequence* (*partial schedule*) will be used throughout this paper. Suppose, for instance, that the problem involves six jobs-labeled J1 through J6. Examples of complete sequences (schedules) include {J2, J5, J1, J3, J6, J4} and {J1, J2, J3, J4, J5, J6}. Again, {J2, J3, J5, J1} is a four-job partial sequence for the same problem. The following are all possible two-job partial sequences involving J1, J4 and J6: {J4, J6}, {J6, J4}, {J1, J4}, {J4, J1}, {J1, J6}, and {J6, J1}.

Heuristics for solving flowshop scheduling problems can be broadly divided into two categories: constructive heuristics and improvement heuristics. A constructive heuristic generates a schedule of jobs in a series of steps, starting first with just a few jobs, adding new jobs to a partial schedule at every step, and eventually arriving at the complete schedule. An improvement heuristic, on the other hand, starts with a (possibly low-quality) complete schedule and improves upon it, by altering it at successive iterations of the algorithm. A third category — a hybrid approach — is also possible, where a mix of constructive and improvement techniques is applied. Hybridization opens up myriad possibilities — two (or more) heuristics (constructive and/or improvement) may be combined in many ways, including combinations of global search and local search algorithms.

Examples of important constructive heuristics (with the total flowtime minimization criterion) include Rajendran and Chaudhuri [8], Rajendran [9], Rajendran and Ziegler [1], Woo and Yim [10], Liu and Reeves [11] and Framinan and Leisten [12]. Liu and Reeves [11] presented and empirically analyzed what they called *composite heuristics* for total flowtime minimization — a bunch of methods where the result produced by a constructive heuristic is further refined by some form of local search.

In this paper, we first propose a constructive heuristic and then develop two composite heuristics that employ a mix of our constructive heuristic, simulated annealing [13] and the Nawaz-Enscore-Ham [3] algorithm. The remainder of the paper is organized as follows: Sect. 2 provides a brief review of related work; Sect. 3 presents the new constructive heuristic that is used in the composite heuristics developed in Sect. 4. Section 5 analyzes empirical results of the proposed schemes and other competing methods. Conclusions are drawn in Sect. 6.

2 Relevant previous work

In the present paper we will need to refer to the following two papers rather frequently: Nawaz et al. [3] (or NEH for short) and Liu and Reeves [11] (or LR). A brief discussion of the main contributions of these two papers is in order.

2.1 NEH heuristic

The NEH algorithm [3] minimizes makespan, not flowtime. However, our constructive heuristic H (Sect. 3) draws inspiration from NEH. Despite the existence of a plethora of flowshop scheduling heuristics for minimizing makespan, NEH continues to be one of the best constructive heuristics because of its simplicity, solution quality and time complexity. In NEH, an initial schedule of jobs — termed the "seed" schedule — is developed by arranging jobs in the descending order of total processing time on all machines. The first two jobs are picked from the seed sequence and the best two-job partial sequence is selected. Inserting the other (unscheduled) jobs from the seed sequence one by one at all possible positions of the current best partial schedule, complete schedules are generated. Finally, the best n-job schedule among the generated schedules is selected.

2.2 Composite heuristics of Liu and Reeves

Liu and Reeves [11] proposed a new constructive heuristic that uses a specially designed index function to choose which job from the list of unscheduled jobs is to be selected to be appended to the already scheduled jobs. The index function consists of the weighted sum of two parts: the total machine idle time and the artificial total flow time. They compared four versions of their new constructive heuristic, namely H(1), H(2), H(n/10) and H(n) (n being the number of jobs) with those of Wang et al. [14], Ho [15], Rajendran and Ziegler [1], and Woo and Yim [10]. Three versions of the constructive heuristic, H(1), H(n/10), H(n), were then combined with six different local search methods to build a set of composite heuristics. Three variants each of two neighborhood schemes, *forward pairwise exchange* and *backward pairwise exchange*, were used in the local search. Thus $3 \times 6 = 18$ types of composite heuristics were obtained. It was empirically shown that the composite heuristics are more effective than the constructive heuristics on Taillard's [16] benchmark problems, but at the cost of additional computation time.

3 A constructive heuristic

The proposed heuristic, H, builds the n-job sequence incrementally, and is therefore a constructive method. It uses the idea of generating partial schedules based on the principle of job insertion of NEH. What leads to its improved performance is its use of a group of promising partial solutions at each stage (i.e., as each new job is added to the sequence). An outline of the method is given below:

1. For each job j, find the total processing time T_j which is given by $T_j = \sum_{i=1}^{m} t_p(i,j),\ j = 1, 2, ..., n$
2. Sort the n jobs on ascending order of their total processing times.
3. Set k=2.
   Take the first two jobs from the sorted list. Find the two 2-job partial sequences and consider the better as the current sequence.
4. Update k=k + 1.
   The k-th job on the sorted list is inserted into k possible positions in the (k-1)-job current sequence, generating k, k-job partial sequences. Select the best (with respect to total flowtime) k-job partial sequence from among those k sequences and set that as the current sequence. Next, each job (except the k-th job of the sorted list) from the k-job current sequence is placed, one by one, into the (k-1) positions of the current sequence. As a result, the (k-1) jobs of the current sequence produce $(k\text{-}1)^2$ partial sequences. Thus the total number of k-job partial sequences in this step is $k+(k\text{-}1)^2$. Determine the best (with respect to the total flowtime) k-job sequence among them and treat it as the current sequence.
5. If k=n, accept the current sequence as the final solution and stop. Otherwise go to Step 4.

4 Two composite heuristics

We propose two composite heuristics, H-1 and H-2, by hybridizing (i) Chakravarthy and Rajendran's [13] version of simulated annealing (SA), (ii) the constructive heuristic H presented in the previous section, and (iii) the classic NEH algorithm [3].

First, the sorted sequence of jobs (based on the ascending order of their total processing times on all machines) is used as the initial (start) sequence for the SA algorithm. The solution generated by the SA is then used as the initial sequence for H-1 or H-2. The SA generates a neighboring sequence from a given sequence by the *adjacent interchange scheme* [13]. The key parameters for SA used in this paper are given in Table 1 (these are the same as the values used in the Chakravarthy-Rajendran paper [13], except for the initial and final temperatures which were 475 and 20, respectively, in that paper; the temperatures are changed in the present study to allow more trial sequences to be generated).

Table 1 Parameter settings for the SA

Parameter	Description	Value
T_0	Initial temperature	675
T_f	Final temperature	1
Fr_cnt	Freeze counter	5
Total	Total number of moves at a particular temperature	$4n$
Per	Percentage of accepted moves	15
r_f	Temperature reduction factor	0.9
Accept	Number of accepted moves at a particular temperature	n

Each of the composite heuristics H-1 and H-2 starts with a sequence produced by the SA and then improves upon it in several iterations. In each of these iterations, NEH is invoked, followed by H. The number of iterations was empirically chosen as five, as a quick-and-dirty compromise between solution quality and the time to find the solution (no attempt was made to find the optimal number of iterations). H-1 and H-2 are similar, differing mainly in how, in each iteration, the output of the NEH algorithm is (or is not) fed as input to H. The pseudocode of H-1 and H-2 is given below. In the pseudocode, a sequence is better than another if the total flowtime of the former is less than that of the latter. For H-1, inside the main loop the sequence produced by NEH is unconditionally used as the start sequence of H (the idea is to allow for the possibility of generating a good solution in future from a not-so-promising current solution). (Recall from Sect. 3 that in the pseudocode for H the first two steps were needed to create the initial sequence of jobs. When used within H-1 or H-2, the code for H would need to be slightly modified: the output of the first two steps in the pseudocode for H is to be replaced with the given sequence, *best-so-far-sequence*.) For H-2, however, the start sequence of H (inside the loop) is the better of NEH's solution and SA's solution.

Composite heuristic #1 (H-1):
best-so-far-sequence ⟵ SA's sequence;
do 5 times
{
current-best-sequence ⟵ best-so-far-sequence;
execute NEH using best-so-far-sequence as the initial sequence;
best-so-far-sequence ⟵ NEH's sequence;
execute H using best-so-far-sequence as the initial sequence;
if
H's sequence is better than the better of current-best-sequence and best-so-far-sequence

Table 2 ARPD and MPD with respect to the best solution

		LR		SA		H - 1		H - 2	
n	*m*	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	0.8221	2.1664	10.009	20.6709	0.0363	0.2166	0.3880	1.2870
	10	0.8788	1.4155	7.0631	12.1770	0.0894	0.4686	0.1244	0.6071
	20	0.5019	1.3648	5.4958	8.1720	0.1445	0.6791	0.2393	1.0163
50	5	0.3471	0.7628	12.0162	16.8939	0.2799	0.6291	0.1133	0.5348
	10	1.0401	1.9278	13.2673	18.2166	0.7358	3.8323	0.2979	0.9899
	20	0.8644	2.2664	10.2807	12.6283	0.1403	0.5337	0.2247	1.2021
100	5	0.1501	0.4289	10.1281	14.3219	0.1332	0.4488	0.1513	0.3452
	10	0.2421	0.9840	12.3532	12.8329	0.1312	0.5494	0.1785	0.6693
	20	0.8412	2.1420	11.2089	14.0005	0.2273	0.9957	0.1121	0.4492

then
best-so-far-sequence ⟵ H's sequence;
else
best-so-far-sequence ⟵ better of NEH's sequence and H's sequence;
}
output best-so-far-sequence;
Composite heuristic #2 (H-2):
best-so-far-sequence ⟵ SA's sequence;
do 5 times
{
current-best-sequence ⟵ best-so-far-sequence;
execute NEH using best-so-far-sequence as the initial sequence;
if
NEH's sequence is better than best-so-far-sequence
then
best-so-far-sequence ⟵ NEH's sequence;
execute H using best-so-far-sequence as the initial sequence;
if
H's sequence is better than best-so-far-sequence
then
best-so-far-sequence ⟵ H's sequence;
if best-so-far-sequence is the same as current-best-sequence (i.e., if neither NEH nor H produced a sequence better than current-best-sequence)
then
best-so-far-sequence ⟵ better of NEH's sequence and H's sequence;
}
output best-so-far-sequence;

5 Experimental results

Three algorithms — the Chakravarthy-Rajendran [13], simulated annealing-based approach (SA) and the composite heuristics H-1 and H-2 — were run on nine different problem sizes (n=20, 50, 100 and m=5, 10, 20) from Taillard's [16] benchmarks. For each problem size, ten independent instances were created following Taillard's function *unif()* and *time seeds* [16]. Thus there are 90 problem instances each of which corresponds to a new t_p matrix.

H-1, H-2, and the SA were coded in C and run on a Pentium 4, 256 MB, 2.8 GHz PC. The proposed methods are compared with the SA and also with the composite

Table 3 Number of cases where best solutions were found and the average (over 10 instances) total flowtime

n	*m*	LR		H-1		H-2	
		No. of best solutions	Mean flowtime	No. of best solutions	Mean flowtime	No. of best solutions	Mean flowtime
20	5	1	14123	7	14012	5	14046
	10	0	20290	7	20133	6	20139
	20	2	33315	6	33198	6	33227
50	5	3	67443	1	67396	6	67285
	10	1	88425	3	88145	6	87757
	20	0	124869	6	123981	4	124086
100	5	4	243228	3	243267	3	243232
	10	3	295772	3	295397	4	295532
	20	2	387303	5	384528	6	384971

heuristics of Liu and Reeves [11]. We did not code the LR heuristics; we used the results published in [11] for our comparative analysis. No single composite heuristic of Liu and Reeves [11] performs best for all the problems in the benchmarks, and in that paper more than one composite heuristic have been shown to produce the best flowtimes in several cases. Therefore, in the present paper, we use the phrase "the composite heuristics of Liu and Reeves" to mean any of the Liu-Reeves composite heuristics that produced the best solution for a particular problem instance. Specifically, the LR flowtimes used in Tables 2, 3, 4, 5 and 6 of the present paper are obtained from the "Best sol. found" column in Table 6 of [11].

Table 2 compares the four methods using the following performance measures: average relative percentage deviation (ARPD) and the maximum percentage deviation (MPD). The ARPD and MPD are defined as follows (TFT stands for total flowtime of jobs, and H represents the heuristic):

$$\mathrm{ARPD} = \frac{100}{10}\sum_{i=1}^{10}\left(\frac{\mathrm{TFT_{H,i}} - \mathrm{BestTFT_i}}{\mathrm{BestTFT_i}}\right)$$

$$\mathrm{MPD} = \max i\left(\frac{\mathrm{TFT_{H,i}} - \mathrm{BestTFT_i}}{\mathrm{BestTFT_i}}\right) \times 100$$

Clearly, the best possible performance corresponds to both ARPD and MPD being zero. The "best TFT" used in the ARPD and MPD calculations is the best among the four solutions. Table 2 shows that the proposed heuristics outperform LR for almost all problem sets. However, for 100 jobs and 5 machines, the performance of H-1 and H-2 is marginally inferior to LR. Table 2 shows that the SA is the

Table 4 Comparison of total flow times for different heuristics (LR = Liu & Reeves (2001), SA = Simulated annealing (Chakravarthy & Rajendran, 1999), H-1 = proposed heuristic 1, H-2 = proposed heuristic 2). The benchmark problems (number of jobs = 20) are due to Taillard (1993)

n	*m*	LR	SA	H-1	Difference		*t*	H-2	Difference		*t*
					Mean	Std. dev.			Mean	Std. dev.	
20	5	14226	15014	14046	110.8	95.95	3.65	14046	58.6	125.82	1.47
		15446	16331	15263				15395			
		13676	16153	13415				13386			
		15750	16706	15567				15731			
		13633	14493	13633				13618			
		13265	14987	13139				13223			
		13774	14853	13628				13628			
		13968	14886	13973				13973			
		14456	16218	14452				14638			
		13036	14300	13006				13006			
20	10	21207	22860	21009	157.8	56.25	8.87	20911	151.3	91.23	5.24
		22927	24302	22824				22848			
		20072	20700	19979				19979			
		18857	20588	18776				18890			
		18939	21059	18817				18773			
		19608	20510	19439				19402			
		18723	19637	18479				18500			
		20504	21580	20303				20303			
		20561	21511	20348				20433			
		21506	22722	21352				21352			
20	20	34119	36233	34202	117.3	179.70	2.06	34284	88.1	209.52	1.33
		31918	34249	31996				32026			
		34552	35845	34479				34479			
		32159	33762	31726				31902			
		34990	36331	34840				34605			
		32734	35369	32788				32697			
		33449	34937	33053				33053			
		32611	33322	32470				32800			
		34084	35540	33961				33961			
		32537	34068	32465				32465			

Table 5 Comparison of total flow times for different heuristics (LR = Liu & Reeves (2001), SA = simulated annealing (Chakravarthy & Rajendran, 1999), H-1=proposed heuristic 1, H-2=proposed heuristic 2)

n	*m*	LR	SA	H-1	Difference		t	H-2	Difference		*t*
					Mean	Std. dev.			Mean	Std. dev.	
50	5	65663	73273	65788	7.0	263.94	0.08	65837	158.2	241.53	2.07
		68664	75706	69096				68848			
		64378	73262	64436				64178			
		69795	77043	69561				69371			
		70841	79000	70649				70340			
		68084	74296	68051				67911			
		67186	74032	67285				67073			
		65582	73522	65625				65625			
		63968	71190	63731				63555			
		70273	81523	69741				70114			
50	10	88770	100734	89257	280.3	1168.49	0.75	89538	668.3	646.88	3.27
		85600	93909	84634				83981			
		82456	93404	81213				81462			
		89356	99892	88408				88368			
		88482	97870	89194				88226			
		89602	99528	91982				88587			
		91422	102477	90842				90353			
		89549	100699	89308				88849			
		88230	102826	86981				87842			
		90787	99552	89632				90365			
50	20	129095	139659	128469	888.3	742.44	3.78	128741	783.0	884.50	2.80
		122094	135740	120610				121269			
		121379	132529	118931				118689			
		124083	133752	124323				123663			
		122158	133542	121652				121786			
		124061	137626	123331				123428			
		126363	139601	125529				127038			
		126317	140434	125080				124688			
		125318	135399	124241				124363			
		127823	137056	127642				127196			

The benchmark problems (number of jobs=50) are due to Taillard (1993)

poorest among the four approaches; this is not unexpected, since the SA, unlike the other three methods, is not a composite procedure. The goal of the present study is not to compare SA with the proposed heuristics. The SA was run (for a time much shorter than the time for H-1 or H-2) just to produce a sequence that can be subsequently used as the initial sequence for both H-1 and H-2. The SA solutions shown in Tables 2, 4, 5 and 6 are the ones that were used as the initial solutions in H-1 and H-2.

Table 3 shows the success rate of the different methods. For each problem instance, the best of the solutions found by LR, H-1 and H-2 is marked as the "best" for that instance, and then the number of the "bests" found by a given algorithm for each 10-instance set is noted. This number provides a measure of the success rate of a given algorithm on a given problem. Note that it is possible for more than one method to find the same "best" for a given instance. Table 3 also shows the mean of the 10 flowtimes (not all of which are the "best") for each problem size. Both the proposed heuristics are superior to LR for all problem sizes except $n=100$, $m=5$.

Tables 4, 5 and 6 present empirical results of total flowtimes for each of the ten instances of each problem, obtained by LR, SA, H-1 and H-2. The data in these tables show that the proposed heuristic H-1 performs better than LR in 66 instances out of 90. H-2 is better than LR in 68 instances out of 90. If the combined result of H-1 and H-2 is considered, the performance improves to 74 instances out of 90 (82.22%).

Tables 4, 5 and 6 also show results of statistical tests of significance for two separate cases (LR versus H-1 and LR versus H-2). Note that the phrase "LR versus H-1" really means "the best algorithm from among the bunch of algorithms collectively called LR versus H-1". (The algorithms that we here call LR may even include heuristics not attributed to Liu-Reeves: there are six "Woo" entries in

Table 6 Comparison of total flow times for different heuristics (LR = Liu & Reeves (2001), SA = simulated annealing (Chakravarthy & Rajendran, 1999), H-1 = proposed heuristic 1, H-2 = proposed heuristic 2)

n	*m*	LR	SA	H-1	Difference		*t*	H-2	Difference		*t*
					Mean	Std. dev.			Mean	Std. dev.	
100	5	256789	287594	257444	−38.9	689.01	−0.18	256409	−4.3	436.50	−0.03
		245609	274501	246145				245935			
		241013	266587	240976				240675			
		231365	252729	230377				231013			
		244016	270662	244877				243783			
		235793	257817	236150				236607			
		243741	271694	243806				243806			
		235171	267917	234353				234787			
		251291	283093	251885				251885			
		247491	270922	246655				247422			
100	10	306375	337234	305696	375.8	1303.75	0.91	306692	242.0	1284.11	0.60
		280928	316073	281272				281422			
		296927	335293	296856				296598			
		309607	341298	309428				308577			
		291731	329427	291728				291728			
		276751	316874	277283				277297			
		288199	320648	286251				287345			
		296130	334357	297757				298112			
		312175	348804	309133				309239			
		298901	330834	298562				298308			
100	20	383865	418032	375815	2775.8	2846.67	3.08	375815	2332.5	3497.06	2.11
		383976	436450	382849				384800			
		383779	416032	380334				380798			
		384854	428088	384583				382863			
		383802	422639	379477				379477			
		387962	423331	382711				381754			
		384839	425737	386101				388671			
		397264	443856	397634				397634			
		387831	428415	384442				386567			
		394861	430781	391329				391329			

The benchmark problems (number of jobs = 100) are due to Taillard (1993)

Table 6 in [11].) Each test suite gives us 10 pairs of flowtime values and we thus have a paired comparison. For each test suite, the mean and the standard deviation of the ten differences in flowtimes are obtained. The difference in each instance is obtained by subtracting the flowtime of the proposed scheme from that of LR. We now test the hypothesis that the population corresponding to the differences has mean, μ, zero. Specifically, we test the (null) hypothesis $\mu=0$ against the alternative $\mu>0$. We assume that the flowtime difference is a normal random variable, and choose the significance level $\alpha=0.5$. If the hypothesis is true, the random variable

$$t = \sqrt{N}\frac{\overline{X} - \mu_0}{S}$$

has a *t*-distribution with N - 1 degrees of freedom, where N = sample size, $\overline{X}$ = sample mean, S = sample standard deviation, and $\mu_0=0$ [17]. The critical value c is obtained from the relation probability ($t>c$) = $\alpha=0.05$. From the standard tables of *t*-distribution, we have for nine degrees of freedom $c=1.83$. For example, the first statistical test (LR versus H-1) in Table 4 corresponds to $N=10$, $\mu_0=0$,

Table 7 Average (over 10 instances) computation time consumed by H-1 or H-2 for a single instance

n	*m*	CPU time (seconds)
20	5	0.2
	10	0.4
	20	0.86
50	5	6
	10	12
	20	25
100	5	99
	10	197
	20	156

$\overline{X}$=110.8, S=95.95, and the sample $t = \sqrt{10}(110.8 - 0)/95.95 = 3.65$. Since $t > 1.83$, we conclude that the difference is statistically significant. From Tables 4, 5 and 6 we see that in most of the cases, H-1 or H-2 or both are statistically significantly better than LR. There is only one case (n=100, m=5) where the mean difference is negative for both H-1 and H-2, but even in this case LR is *not* statistically significantly better than our approach.

To provide an idea of the amount of time involved in the computation, Table 7 shows the execution time for a single instance of the problem (obtained as the average of ten independent instances). No attempt at refining or optimizing the code was made, and debugging, print and other statements were not deleted. The times for H-1 and H-2 were approximately the same. For 100 jobs and 20 machines (the last row in Table 7), the main loop was executed only two times, instead of five (hence the relatively shorter time). As can be seen from the pseudocode for H-1 and H-2, the major part of both of these algorithms is spent in the outermost loop, and the NEH algorithm accounts for a substantial part of the loop cost. The experiments reported here used the original NEH algorithm which is known to have a time complexity of $O(mn^3)$. A better variant of NEH, such as Taillard's [18] improvement of NEH that runs in $O(mn^2)$ time, would result in significant savings in the run time.

6 Conclusion

Two composite heuristics for the minimization of total flowtime in permutation flowshop scheduling problems have been presented in this paper. Computational experiments on standard benchmark problems have been carried out. The composite heuristics have been shown to produce results that meet or beat, in a statistically meaningful way, those produced by the composite heuristics of Liu and Reeves [11]. This study shows that composite (or hybrid) heuristics are an attractive alternative to the more traditional simple heuristics for flowshop problems, especially when the solution quality is of prime concern. A downside of complicated composite procedures is the larger computation times. However, with computationally efficient versions of the basic algorithms that build the composite method, the run time of composite procedures can be brought down to acceptable levels.

A difference between the proposed algorithms and the composite heuristics of [11] is that the latter approach is rather problem specific in the sense that no single heuristic produced high-quality solutions for all the test problems, whereas either of the two composite heuristics presented in this paper performed very well on almost all the problems in the testbed.

Acknowledgments We are grateful to three anonymous referees for helpful comments on an earlier version.

References

1. Rajendran C, Ziegler H (1997) An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. Eur J Oper Res 103:129–138
2. Framinan JM, Leisten R, Ruiz-Usano R (2005) Comparison of heuristics for flow time minimization in permutation flow shop. Comput Oper Res 32:1237–1254
3. Nawaz ME, Enscore E, Ham I (1983) A heuristic algorithm for the m- machine, n-job flowshop sequencing problem. Omega 11:91–95
4. Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155:426–438
5. Laha D, Chakraborty UK (2007) An efficient stochastic hybrid heuristic for flowshop scheduling. Engineering Applications of Artificial Intelligence 20:851–856
6. Chakraborty UK, Laha D (2007) An improved heuristic for permutation flowshop scheduling. Int J Inf Commun Technol 1:89–97
7. Gonzalez T, Sahni S (1978) Flow shop and job shop scheduling: complexity & approximation. Oper Res 26:36–52
8. Rajendran C, Chaudhuri D (1991) An efficient heuristic approach to the scheduling of jobs in a flowshop. Eur J Oper Res 61:318–325
9. Rajendran C (1993) Heuristic algorithm for scheduling in flowshop to minimize total flow time. Int J Prod Econ 29:65–73
10. Woo DS, Yim HS (1998) A heuristic algorithm for mean flow time objective in flowshop scheduling. Comput Oper Res 25:175–182
11. Liu J, Reeves CR (2001) Constructive and composite heuristics solution of the $P||\sum C_i$ scheduling problem. Eur J Oper Res 132:439–452
12. Framinan JM, Leisten R (2003) An efficient constructive heuristic for flowtime minimization in permutation flowshops. Omega 31:311–317
13. Chakravarthy K, Rajendran C (1999) A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. Prod Plan Control 10:707–714
14. Wang C, Chu C, Proth JM (1997) Heuristic approaches for n/m/F/$\sum C_i$ scheduling problems. Eur J Oper Res 96:636–644
15. Ho JC (1995) Flowshop sequencing with mean flow time objective. Eur J Oper Res 81:571–578
16. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
17. Kreyszig E (1972) Advanced engineering mathematics. John Wiley, New York
18. Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47:65–74