An Improvement of Total Flowtime and Makespan for the General Flow Shop Scheduling Problem Via $O(n^4m)$ Algorithm

Sawat Pararach

Department of Industrial Engineering, Faculty of Engineering,
Thammasat University, Pathum Thani 12121
E-mail: psawat@engr.tu.ac.th

Abstract

In general flowshop scheduling problems, *n* jobs are arranged on *m* machines in a series and follow the same routing. To obtain the best schedule, minimizing the flowtime and makespan in a general flow shop has been demonstrated. When the number of job little increases, the computational time to find the best schedule is time consuming or non-polynomial hard (NP-hard) in complexity. This study proposes a polynomial time heuristic, which is complex in $O(n^4m)$, to obtain the solution. From experimental instances (640), the results show that the proposed algorithm yields better performance in terms of the average relative percentage deviation in both flowtime and makespan values than the well-known Framinan and Leisten method, while its time complexity is the same.

Keywords: Flowshop, Total flowtime, heuristic

1. Introduction

For a general flowshop, multi-operation processing concerns a set of *n* jobs and *m* machines. All the jobs are arranged and feed into a fixed sequence of machines. For each machine, the jobs are performed in their processing time. The obtained solution is the schedule that minimizes the total flow time of all jobs. The total flow time is not the makespan objective, but it represents the difference between completion time and release time for each job. The advantage of a total flow time objective is minimization of work-in-process, but is not restricted to no inter-mediate storage inventory. In the case of the makespan objective, it considers the routing that minimizes the completion time of the last job in the last machine, but does not determine the waiting time for each job in the process. The sequence that minimizes total flow time may not be the same as the minimized-makespan sequence.

The Solution for scheduling of *n* jobs in flow shop relies on a set of restrictive assumptions as follows [3]:

-Single part or batch of parts is always treated as a single job,

-Preemption and job cancellation are not allowed.

-Processing times are independent of the schedule,

-Work-in-process is allowed,

-Machines are able to process one job at a time,

-Each job visits all machines exactly once,

-Machines are always available and the only resource modeled,

-Jobs are all known in advance,

-The scheduling is purely deterministic.

Flow shop scheduling has been proved to be NP-hard [5]. For several years, many heuristics for solving these problem have been considered. Averbakh [2] studied the flow-shop problem with two jobs and *m* machines, and the uncertainly interval processing times of operations. Soukhal et. al [9] investigated two-machine flow shop scheduling problems taking transportation into account. Koulamas and Kypasiris [6] studied the two-stage assembly flow shop scheduling with concurrent operations in the first stage and a single assembly operation in the second stage. Yokoyama [10] considered a flow shop scheduling model with partition machining setups, and assembly operations into blocks.

On the improved-heuristics for total flow time minimization, Agarwal et al. [1] developed a non-polynomial time heuristic based on the adaptive learning approach. For polynomial time methods, Framinan and Leisten [4] developed a constructive heuristic based on a pairwise interchange approach, Laha and Sarin [7] improved its performance by node-insertion procedures.

The proposed procedure in this paper is improving the method of Laha and Sarin [7] while not affecting the time-complexity of the $O(n^4m)$ algorithm. The investigation is presented in Section 2. The proposed procedure is described in Section 3. Results of the experimentation are demonstrated in Section 4. Finally, concluding remarks are made in Section 5.

2. The method of Framinan and Leisten and its modification by Laha and Sarin

The first concept for an $O(n^4m)$ polynomial time algorithm was developed from the NEH heuristic method of Nawaz et al. [8] by Framinan and Leisten [4] for solving the minimium total flow time in a permutation flow shop. From the literature the steps are given below:

Step 1: For each job *i*, find the total processing times, T_i on all machines,

$$T_i = \sum_{j=1}^{m} t_{ij} \quad \text{for all } i=1,2,\ldots,n.$$

Step 2: Sort the jobs in ascending order of their total processing time.

Step 3: Set k=2. Select the first two jobs from the sorted list and select the better between the two possible sequences.

Step 4: Increment k, $k=k+1$. Select the k^{th} job from the sorted list and insert it into k possible positions of the best partial sequence obtained so far. Among the k sequences, the best k-job partial sequence is selected based on minimum total flow time. Then, investigate all possible sequences by interchanging the job in position i and j of the above partial sequence for all i,j $(1 \le i < k, i < j \le k)$ and select the best partial sequence among $k(k-1)/2$ sequences having minimum total flow time.

Step 5: If $k=n$, then STOP; else go to step 4.

Sorting the jobs from total processing times

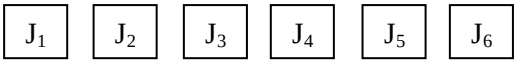


Figure 1. Example of sorting jobs in step 1 and 2 of the method of Framinan and Leisten.

Select the better sequence from the first two jobs

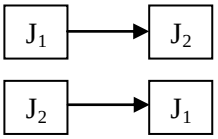


Figure 2: Example of selecting the starting sequence in the step 3 of the method of Framinan and Leisten.

Select the better on the insertion of jobs from the list

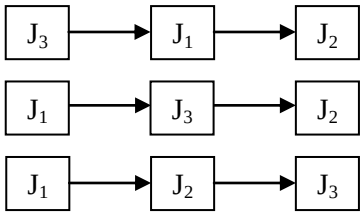


Figure 3. Example of selecting job into the current sequence of the method of Framinan and Leisten.

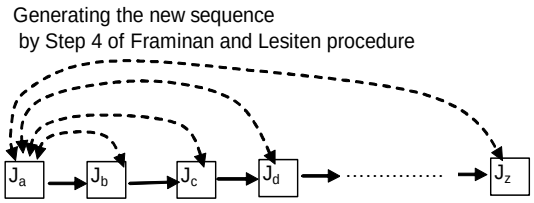


Figure 4: Example of Step 4 of the method of Framinan and Leisten.

Because of the pairwise interchange performed on the k schedules, this rises to a total $k(k\text{-}1)/2$ iterations. From placing the k^{th} job of the sorted list, it requires k iterations. Therefore, the operations are performed $k+[k(k\text{-}1)/2] = k(k+1)/2$. The total flow time for each schedule of k jobs on m machines provides $O(km)$. For all k in Step 4, the overall iteratious of the method of Framinan and Leisten [4] is $k*km*k(k+1)/2$. Replacing k with n, the time-complexity is $O(n^4m)$.

Laha and Sarin [7] have modified Step 4 of the Framinan and Leisten procedure by implementing an insertion step rather than performing pairwise interchanges. Step 4 is modified as:

Step 4: For k=3 to n, do the following.

Insert the k^{th} job on the sorted list into k possible positions of the (k-1)-job current sequence, thereby generating k, k-job partial sequences, and select from these a k-job partial sequence with the best total flow time value. Designate this as a k-job current sequence. Place each job (except for the k^{th} job of the sorted list) of this sequence into its (k-1) positions and select the best k-job sequence having the least total flow time value from among those generated. This becomes the next k-job current sequence.

The calculation to determine the (k-1)-job into the (k-1)-position requires $(k\text{-}1)^2$ iterations and requires k iterations for placing the k^{th} job from the sorted list into the k positions of the (k-1)-job current sequence. Therefore, the operation is performed $k+(k\text{-}1)^2$ iterations.When the $k+(k\text{-}1)^2$ operations replaces the $k(k+1)/2$ operations of the Framina and Leisten method, the modified Step 4 by Laha and Sarin method is $k*km*[k+(k\text{-}1)^2]$. Replace k with n, giving the time-complexity of $O(n^4m)$.

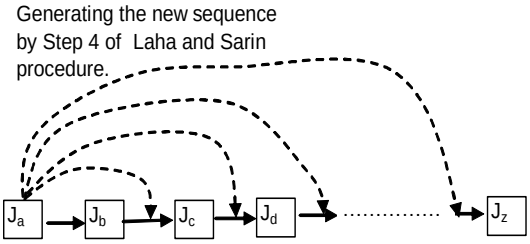


Figure 5: Example of Step 4 of the method of Laha and Sarin.

For the performance evaluation between Laha-Sarin method and Framinan-Leisten method, the experimental report from Laha and Sarin [7] shows that the flowtime values of the Laha-Sarin method outperforms the method of Framinan and Leisten statistically better at α=0.05 . The average CPU time value obtained from Laha-Sarin method takes slightly more time than (the obtained CPU time value) the method of Framinan and Leisten.

3. The proposed $O(n^4m)$ polynomial time algorithm.

From the literature, the effective heuristic has been solved for many general flow shop instances in the time-complexity of $O(n^4m)$ considered by Laha and Sarin [9].

This concept is based on adding the pairwise interchanges of Framinan and Leisten[4] into the obtained sequence of Step 5 of Laha and Sarin[7]. The steps of this concept are given as:

Step 1: For each job i, find the total processing times, T_i on all machines,

$$T_i = \sum_{j=1}^{m} t_{ij} \quad \text{for all i=1,2,…,n.}$$

Step 2: Sort the jobs in ascending order of their total processing time.

Step 3: Set k=2. Select the first two jobs from the sorted list and select the better between the two possible sequences.

Step 4: For k=3 to n do the following.

Insert the k^{th} job on the sorted list into k possible positions of the k-1 job current sequence, thereby generating k, k-job partial sequences, and then selecting from these a k-job partial sequence with the best total flow time value. Then, this sequence is designated as a k-job current sequence. Each job (except for the k^{th} job of the sorted list) of this sequence is placed into its (k-1) position and the best k-job sequence having the least objective value (total flow time or makespan) is selected from among those generated. This sequence becomes the next k-job current sequence.

Step 5: If k=n, then go to Step 6; else, go to Step 4.

Step 6: Interchane jobs in position i and j for all i,j ,$1 \leq i < n,\ i < j \leq n$. Select the best sequence obtained among the $n(n$-1)/2 sequences having minimized objective value.

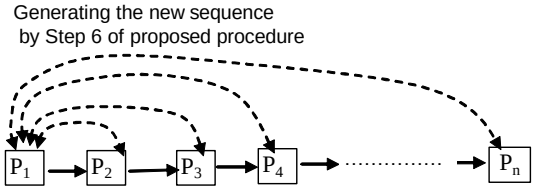


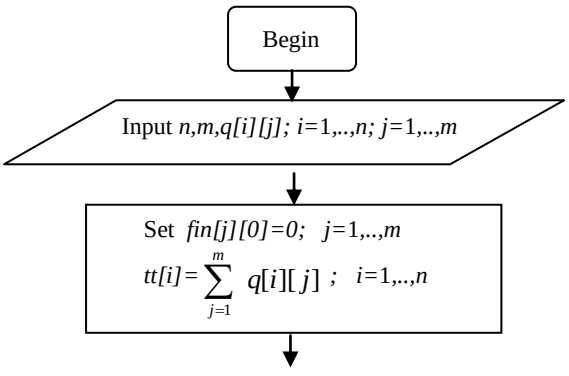
Figure 6: Example of pairwise interchanges in Step 6.

From this procedure, Steps 1-4 are repeated from Steps 1-4 of Laha and Sarin [7]. The modification in this research is the adding of a procedure as Step 6. Step 6 determines the pairwise interchanges on the n-job sequence obtained from STOP mode in Step 5 of Laha and Sarin [7] by interchanging jobs in position i and j for all i,j ,$1 \leq i < n,\ i < j \leq n$. Select the best sequence obtained among the $n(n$-1)/2 sequences having minimum total flow time.

The flowchart of the code program can be described in Figure 7.

The following is a summary of notation that is used in the flowchart.

N	total number of jobs
m	total number of machines
q[i][j]	processing time at job *i* on machine *j*
tt[i]	summation of processing time of job *i* for all machine *j*=1,…,*m*
fin[j][i]	completion time of job *i* on machine *j*



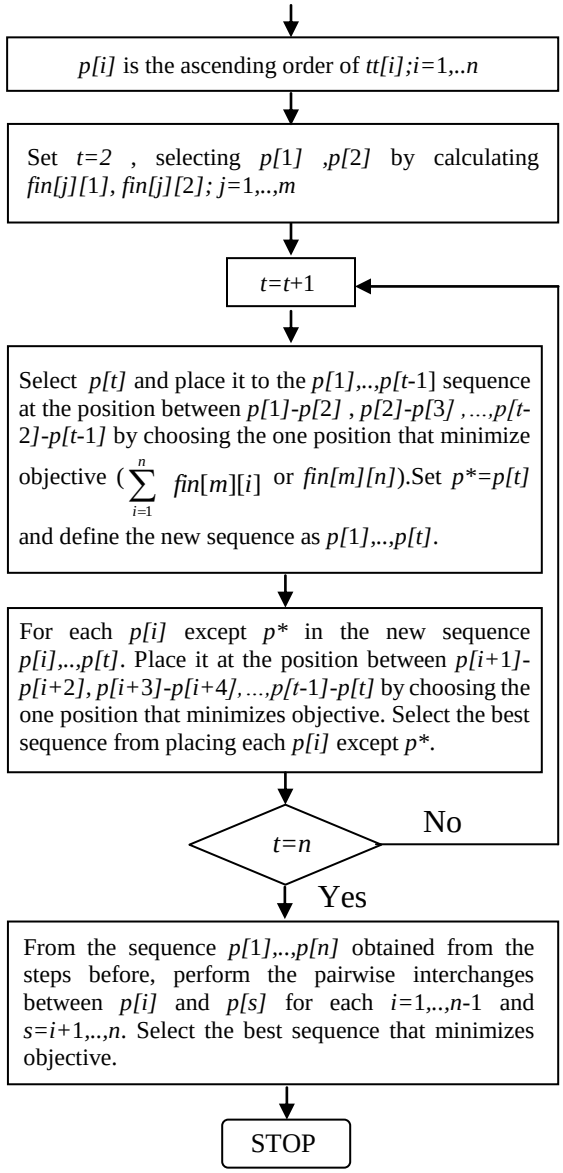


Figure 7. The flowchart of the code program of the proposed method.

Table 1. Comparison of the proposed methods for the objective of minimizing total flow time.

n	m	no.instances	Average CPU times (s)		ARPD
			Laha-Sarin [7]	Proposed	
10	5	1 to 20	0.02655	0.0289	0.982219
10	10	21 to 40	0.02425	0.0266	0.676018
10	15	41 to 60	0.0281	0.0289	0.633287
10	20	61 to 80	0.0282	0.0282	0.535631
20	5	81 to 100	0.1781	0.2492	1.042264
20	10	101 to 120	0.1751	0.3469	0.624009
20	15	121 to 140	0.18045	0.24765	0.669467

Table 1. Comparison of the proposed methods for the objective of minimizing total flow time. (Continued)

n	m	no.instances	Average CPU times (s)		ARPD
			Laha-Sarin [7]	Proposed	
20	20	141 to 160	0.18985	0.19535	0.567634
30	5	161 to 180	0.58365	0.5867	0.766617
30	10	181 to 200	0.6273	0.63125	0.54204
30	15	201 to 220	0.6187	0.62575	0.449736
30	20	221 to 240	0.6479	0.65705	0.447026
40	5	241 to 260	1.4024	1.40705	0.662283
40	10	261 to 280	1.46015	1.4656	0.559147
40	15	281 to 300	1.5196	1.5297	0.359461
40	20	301 to 320	1.58115	1.5984	0.40412
50	5	321 to 340	2.79305	2.80235	0.565697
50	10	341 to 360	2.93995	2.95235	0.566107
50	15	361 to 380	3.0968	3.1172	0.374524
50	20	381 to 400	3.2414	3.2719	0.384409
60	5	401 to 420	4.9523	4.96715	0.678612
60	10	421 to 440	5.31495	5.33595	0.428656
60	15	441 to 460	5.6181	5.65235	0.386263
60	20	461 to 480	5.91245	5.9539	0.336179
70	5	481 to 500	8.229	8.2469	0.477944
70	10	501 to 520	8.72655	8.7656	0.357064
70	15	521 to 540	9.3578	9.4156	0.391865
70	20	541 to 560	9.87975	9.9476	0.403442
80	5	561 to 580	12.48355	12.51095	0.606884
80	10	581 to 600	13.5281	13.58675	0.414618
80	15	601 to 620	14.50235	14.5867	0.39081
80	20	621 to 640	15.46335	15.5617	0.315106

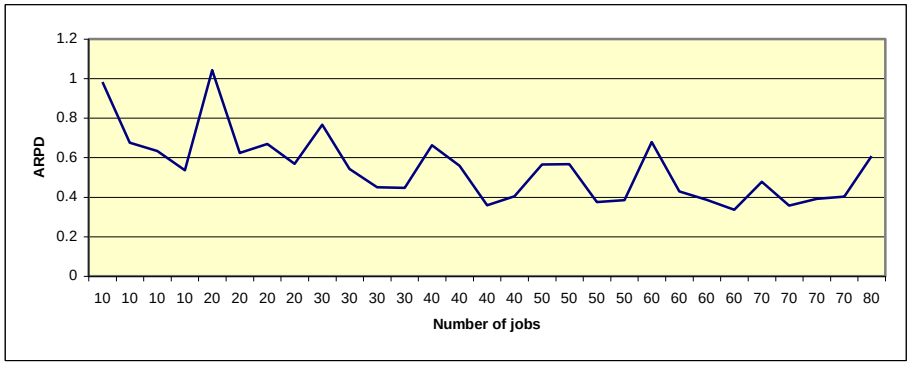


Figure 8. ARPD and size of problem for minimizing total flow time objective.

Table 2. Comparison of the proposed methods for the objective of minimizing makespan.

n	m	no.instances	Average CPU times (s)		ARPD
			Laha-Sarin [7]	Proposed	
10	5	1 to 20	0.0398	0.0406	0.550132
10	10	21 to 40	0.0218	0.04295	0.851801
10	15	41 to 60	0.04065	0.04225	1.311459
10	20	61 to 80	0.03905	0.0414	0.823365
20	5	81 to 100	0.1679	0.1703	0.874063
20	10	101 to 120	0.1812	0.18745	1.037284
20	15	121 to 140	0.19545	0.1977	0.575118
20	20	141 to 160	0.19995	0.2023	0.830651
30	5	161 to 180	0.4578	0.45855	0.934585
30	10	181 to 200	0.5109	0.51715	0.806254
30	15	201 to 220	2.2564	0.53755	0.565922
30	20	221 to 240	0.54455	0.5492	0.527077
40	5	241 to 260	1.02025	1.02185	0.503105
40	10	261 to 280	1.11785	1.125	0.650253
40	15	281 to 300	1.15305	1.16165	0.521265
40	20	301 to 320	1.2298	1.2392	0.497326
50	5	321 to 340	1.58505	1.9374	0.359122
50	10	341 to 360	2.0976	2.11345	0.456082
50	15	361 to 380	2.25395	2.268	0.432683
50	20	381 to 400	2.3723	2.2852	0.368986
60	5	401 to 420	3.3446	3.35395	0.421359
60	10	421 to 440	3.64675	3.66255	0.38288
60	15	441 to 460	3.90225	3.9258	0.412617
60	20	461 to 480	4.12505	4.15545	0.425646
70	5	481 to 500	5.37105	5.3797	0.371372
70	10	501 to 520	5.87115	5.893	0.302847
70	15	521 to 540	6.32025	6.35625	0.330696
70	20	541 to 560	6.74615	6.79295	0.347304
80	5	561 to 580	8.2007	8.2164	0.260466
80	10	581 to 600	8.95775	8.98995	0.384264
80	15	601 to 620	9.73435	9.7819	0.37028
80	20	621 to 640	10.44765	10.51395	0.301534

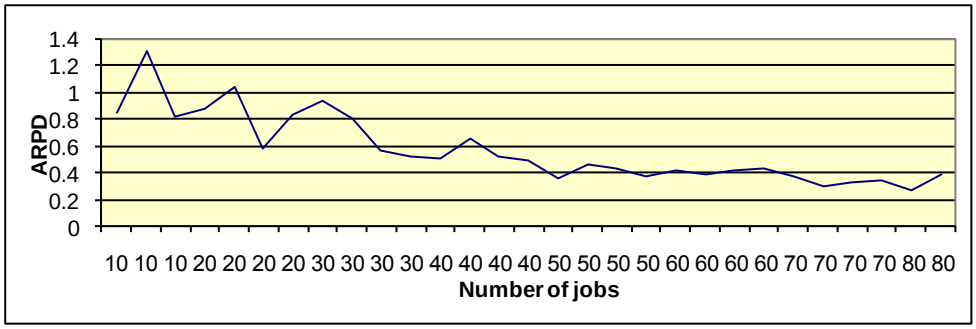


Figure 9. ARPD and size of problem for minimizing makespan objective

p[i] a job acceptable at position i in the sequence.
i index for jobs (i=1,..,n)
j index for machines (j=1,..,m)

Note that Step 6 dictates the time complexity of $O(nm)$ for the schedule of n jobs on m machines. Multiplication of the $O(nm)$ of the schedule by the $O(n^2)$ operations of changing the sequence, the overall executed time in Step 6 is $O(n^3m)$.

From the working paper[7], Step 1 to Step 5 perform $O(n^4m)$ operations. With Step 6 adding to Step 1-5, the procedure performs $O(n^4m)+ O(n^3m) \approx O(n^4m)$ operations , since the proposed procedure in Step 6 does not increase the time complexity of the Laha and Sarin's method.

4. Performance evaluation

The experimentation has been carried out on 640 instances with n=10, 20, 30, 40, 50, 60, 70 and 80, and m=5, 10, 15, and 20, and the replication is 20 for each combination of jobs and machines [7]. For the generated random processing times, it follows a discrete uniform distribution between 1 and 99.

The computer programs of the proposed procedure and the method of Laha and Sarin are coded in C++ language and run on a Pentium 4, 256 MB, 2.4GHz PC.

Average relative percentage deviation (ARPD) is considered to compare the performance of these methods. It is defined as [7]:

$$ARPD = \frac{100}{20}\sum_{i=1}^{20}\frac{A_i - B_i}{B_i}$$

For the i^{th} instance, A_i is the objective value obtained from the Laha and Sarin's method and B_i is the objective value obtained from the proposed procedure. The results are demonstrated in **Table 1.**

For all instances, the objective of minimizing total flow time and minimizing makespan can be applied to the code program for comparing 2 methods.

From the results, it is evident that the proposed procedure gives solution values better than the obtained value from the Laha and Sarin's method for the objective of minimizing total flow time and minimizing makespan by ARPD and is greater than 0 for all cases. The better results from the objective of minimizing total flow time and minimizing makespan have been shown as ARPD in Table 1 and Table 2, Figure 8, and Figure 9 respectively.

The computing times of the proposed procedure are greater than the Laha and Sarin's method, but the differences are very small (they are not greater than 0.1 second).

5. Conclusions

This paper studies a minor change on existing heuristic by adding a step of pairwise interchanging into the heuristic Laha and Sarin solution in order to enhance the performance measurement of total flow time and makespan value. The new procedure requires a time-complexity of $O(n^4m)$. The results show that the minor modifications on the heuristic create a significant improvement in the performance as can be seen by the Average relative percentage deviation (ARPD) values. However, the computational time is slightly increased.

6. References

[1] Agarwal A., Colak S., and Eryasoys E., Improvement Heuristic for the Flow-shop Scheduling Problem: An Adaptive Learning Approach, European Journal of Operational Research, Vol. 169, pp. 801-815, 2006.

[2] Averbakh I. The Minmax Regret Permutation Flow Shop problem with

two jobs, European Journal of Operational Research, Vol. 169, pp. 761-766, 2006.

[3] Brandimarte P., and Villa A., Advanced Models for the Manufacturing Systems Management, U.S.A, CRC Press, 1992.

[4] Framinan J.M., and Leisten R., An efficient Constructive Heuristic for Flowtime Minimization in Permutation Flowshops, Omega, Vol. 31, pp .311-317, 2003.

[5] Gonzalez T., and Sahni S., Flow Shop and Job Shop Scheduling: Complexity and Approximation, Operations Research, Vol. 26, pp. 36-52, 1978.

[6] Koulamas C., and Kyparisis G., A Note on the Two-stage Assembly Flow Shop Scheduling Problem with Uniform Parallel Machines, European Journal of Operational Research, Vol. 182, pp. 945-951, 2005.

[7] Laha D., and Sarin S.C., A Heuristic to Minimize total Flow Time in Permutation Flow Shop, Omega, Vol. 37, pp. 734-739, 2009.

[8] Nawaz ME., Enscore E., and Ham I., A Heuristic Algorithm for the M-machine, n-job Flow Shop Sequencing Problem, Omega, Vol. 11, pp. 91-95, 1983.

[9] Soukhal A., Oulamara A. and Martineau P. Complexity of Flow Shop Scheduling Problems with Transportation Constraints, European Journal of Operational Research, Vol. 161, pp. 32-41, 2005.

[10] Yokoyama M., Flow-shop Scheduling with Setup and Assembly Operations, European Journal of Operational Research, Vol. 187, pp. 1184-1195, 2008.