Study on Heuristics for the Permutation Flowshop with Sequence Dependent Setup Times

Xingye Dong, Houkuan Huang and Ping Chen
School of Computer and IT, Beijing Jiaotong University, 100044 Beijing, China
xydong@bjtu.edu.cn, hkhuang@bjtu.edu.cn and chenpingbjtu@gmail.com

Abstract

It is studied that several constructive heuristics for solving the sequence dependent setup time flowshop problem with the objective of minimizing makespan. Three priority rules imbedded in the heuristics are tested and a tie-breaking strategy is examined. The experimental results on benchmarks show that the priority rules are helpful to improve the performance, especially for the instances in which setup times are averagely smaller than the average processing time. The results also show that the setup times have a large effect on the performance of the heuristics.

keywords: Scheduling, Heuristic, Flowshop, Sequence dependent setup time, Makespan

1. Introduction

In the regular permutation flowshop sequencing problem, a set of jobs $J = \{1, 2, \ldots, n\}$ available at time zero needs to be processed on m machines, where $n \geq 1$ and $m \geq 1$. Each job has m operations, each of which needs an uninterrupted processing time. The processing time of the ith operation of job j is denoted by p_{ij}, where $p_{ij} \geq 0$. The process sequence of these m operations is that the first operation needs to be processed on the first machine, and the second operation needs to be processed on the second machine, etc. An operation of a job starts to be processed only if the previous operation of the job is completed and the requested machine is available. Each machine processes these jobs in the same order and only one operation can be processed at a time.

In the above problem, setup time is assumed negligible or part of the job processing time. Though this assumption simplifies the analysis and it really reflects certain applications, there are many scheduling problems requiring an explicit treatment of setup times, i.e., setup times are dependent on the processing sequence of the jobs. This problem is called Sequence Dependent Setup Time Flowshop Problem (SDST flowshop problem). In this problem, it is denoted by s_{ijk} the deterministic and non-negative setup time on machine i when job k follows job j. The objective studied in this paper is to minimize the makespan (denoted by C_{max}), i.e. the time of the last job completed on machine m, and it can be computed in the following way:

Let π denote a permutation, which represents a job processing order on the set J. Let $\pi(k)$, $k = 1, \ldots, n$, denote the kth job in π, then the completion time of $\pi(k)$ on machine i can be denoted by $C_{i,\pi(k)}$ and computed by Eq. (1):

$$C_{i,\pi(k)} = max\{C_{i-1,\pi(k)}, C_{i,\pi(k-1)} + s_{i\pi(k-1)\pi(k)}\} + p_{i\pi(k)} \quad (1)$$

where $C_{i,\pi(0)} = C_{0,\pi(j)} = 0$ for all i and j. This problem can be represented as $F_m|prmu, s_{ijk}|C_{max}$ [7] and is known to be NP-complete for any $m \geq 1$ [4, 5].

In the literature, the methods on this problem can be classified into three categories: exact methods, improvement methods and constructive methods. Exact methods are mainly integer programming methods, e.g. Stafford and Tseng [15], or branch and bound methods, e.g. Ríos-Mercado and Bard [10]. These methods can only solve small size instances, e.g. less than 9 jobs and 9 machines. Improvement methods are mainly meta-heuristics, such as genetic algorithms [12] and iterated greedy algorithms [13]. These methods are often time-consuming. For more information on these two kinds of methods, the readers are referenced to the comprehensive survey by Allahverdi et al [1]. In this paper, we only focus on constructive algorithms.

As for constructive algorithms on the SDST flowshop, the work is scarce. Relatively early work is mainly on the two-machine problem, e.g. Gupta and Darrow [5]. For the general SDST flowshop problem, Simons proposed two heuristics named SETUP and TOTAL [14]; Das et al proposed a heuristic based on a savings index (SI) [2]; Ríos-Mercado and Bard modified the famous NEH [6] heuristic, which was proposed for solving the regular permutation flowshop problem, and developed the NEHT-RB heuristic [8]; later they proposed the HYBRID heuristic [9] based on the work of Simons [14]; recently, Tseng et al [18] pro-

978-1-4244-4116-7/09/$25.00

posed a penalty-based heuristic (PB) based on the work of Das et al [2], which showed better performance than the heuristic by Das et al when $m \geq 4$. The work of Ruiz et al [11] showed that the NEHT-RB by Ríos-Mercado and Bard [8] performed the best among the SETUP and TOTAL heuristics by Simons [14], the SI heuristic by Das et al [2] and the HYBRID heuristic by Ríos-Mercado and Bard [9] on four different sequence dependent Taillard-based benchmarks. To the best of our knowledge, there is no comparison between the NEHT-RB [8] heuristic and the PB heuristic [18] at present.

In the literature, the NEH heuristic [6] was considered as the best constructive algorithm for solving the regular permutation flowshop problem [11]. In our previous work [3], this heuristic is improved by developing a more effective priority rule and a tie-breaking strategy. Since the NEHT-RB heuristic [8] is based on the NEH heuristic, it is worthy examining the effectiveness of these methods on the NEHT-RB heuristic.

The remainder of this paper is organized as follows. In Section 2, the NEHT-RB heuristic [8] is briefly illustrated and compared with the PB heuristic [18]. In Section 3, three priority rules are analyzed, and a tie-breaking strategy is examined in Section 4, which are aimed to improve the NEHT-RB heuristic [8]. Then the paper is concluded in Section 5.

2 NEHT-RB Heuristic

As aforementioned, the NEHT-RB heuristic [8] extends the NEH heuristic [6] to handle setup times. The NEH heuristic was originally proposed for the regular permutation flowshop problem and its time complexity is $O(mn^2)$ by using a speed-up method [16]. The speed-up method can also be used in the NEHT-RB heuristic, and so the time complexity of NEHT-RB is also $O(mn^2)$. The NEHT-RB heuristic can be presented as follows:

(1) sort the n jobs by decreasing sums of processing times on the m machines (setup times are not considered);

(2) take the first two jobs and schedule them in order to minimize the partial makespan as if there were only two jobs (setup times are considered);

(3) for the kth job, $k = 3, \ldots, n$, insert it into the place, among k possible ones, which minimizes the partial makespan (setup times are considered).

It is easy to see that the NEHT-RB heuristic consists of two phases: firstly, the jobs are sorted by descending sums of their processing times (setup times are not considered); secondly, a job sequence is constructed by evaluating the partial schedules originating from the initial order of the previous phase. In our implementation, the heap sort algorithm is used in the first phase. While in the second phase, a job is always inserted into the first position where the minimum makespan is achieved.

As previously stated, the NEHT-RB was shown the best among several constructive algorithms on a set of Taillard-based benchmarks [11]. However, there is no performance comparison with the PB heuristic proposed by Tseng et al [18] at present. In this section, a comparison is made between these two heuristics on the above benchmarks.

In order to describe the benchmarks, we firstly introduce the Taillard benchmarks [17]: the set of Taillard benchmarks includes 120 instances and is divided into 12 groups, each with 10 instances, according to different numbers of jobs and machines; the smallest instances have 20 jobs and 5 machines while the largest instances have 500 jobs and 20 machines; the processing times of all the operations are generated uniformly from [1, 99]. The SDST Taillard-based benchmarks are generated by adding sequence dependent setup times to the above Taillard benchmarks, where the maximum possible setup time is 10%, 50%, 100% and 125% of the maximum possible processing time, respectively, i.e. the setup times are generated uniformly from [1, 9], [1, 49], [1, 99] and [1, 124] respectively. Thus there are total of 480 instances divided into four groups, which are denoted by SDST10, SDST50, SDST100 and SDST125 respectively. The benchmarks can be downloaded from `http://www.upv.es/gio/rruiz`.

The comparison was done on a PC with an Intel Core 2 Duo CPU (2.66GHz) and 1.96G main memory. The results are shown in Table 1 in terms of average relative percentage deviation (RPD). The RPDs are calculated by Eq. (2).

$$\mathrm{RPD} = (C_{max}^{PB} - C_{max}^{NEHT-RB})/C_{max}^{NEHT-RB} \times 100\% \quad (2)$$

where C_{max}^{PB} denotes the makespan obtained by the PB heuristic, and $C_{max}^{NEHT-RB}$ denotes the makespan obtained by the NEHT-RB heuristic.

From Table 1, it can be seen that the PB heuristic performs worse than the NEHT-RB on the SDST10 and SDST50 benchmarks for the average RPDs for all instances are 4.75 and 2.06 respectively, while it performs better on the SDST100 and SDST125 benchmarks and the average RPDs for all instances are -1.26 and -2.30 respectively. This shows that the PB heuristic performs better on the instances with large setup times. This is in accordance with the claim of its authors [18]. According to Gupta and Darrow [5], setup times of many real-world instances are narrow, just like the SDST10 and SDST50 benchmarks. So it is meaningful to improve the performance of NEHT-RB, especially on the SDST10 and SDST50 benchmarks.

When the CPU time is considered, the NEHT-RB needs much less CPU time for its time complexity is $O(mn^2)$, while the PB needs much more CPU time for its time complexity is $O(mn^3)$ [18]. We list the CPU times on SDST10 benchmarks in Table 2 in seconds, and the CPU times on SDST50, SDST100 and SDST125 are omitted for they are very similar to those listed in this table.

Table 1. RPD comparison between NEHT-RB and PB heuristic

	SDST10	SDST50	SDST100	SDST125
20|5	3.86	2.49	-2.73	-1.51
20|10	4.17	1.06	0.48	0.48
20|20	5.15	3.50	2.04	1.84
50|5	3.21	1.21	-2.64	-4.16
50|10	4.14	3.26	-0.06	-1.69
50|20	6.05	3.68	1.49	-0.41
100|5	2.22	-0.70	-4.50	-6.52
100|10	4.55	1.84	-1.87	-3.00
100|20	5.76	2.84	-0.14	-1.15
200|10	3.10	0.14	-3.83	-5.10
200|20	7.00	3.25	-0.66	-2.07
500|20	7.78	2.19	-2.75	-4.34
all	4.75	2.06	-1.26	-2.30

Table 2. CPU times (in seconds) needed by NEHT-RB and PB heuristic on SDST10

	NEHT-RB	PB
20|5	0.0	0.0
20|10	0.0	0.0
20|20	0.0	0.0
50|5	0.0	0.0
50|10	0.0	0.0
50|20	0.0	0.0
100|5	0.0	0.1
100|10	0.0	0.1
100|20	0.1	0.2
200|10	0.1	0.7
200|20	0.3	1.5
500|20	1.7	23.9
all	0.2	2.2

3 Priority Rule

In the first phase of the original NEHT-RB heuristic, the priority rule consists in sorting the n jobs by decreasing sums of processing times on the m machines. This priority rule is the same as that in the NEH heuristic [6]. In the literature, it has been shown that a better priority rule can be obtained by considering the standard deviations of the jobs' processing times at the same time for the NEH heuristic [3], so an improved performance is to be expected for the NEHT-RB heuristic if these standard deviations are included in this procedure. Also, the setup times should be considered in the priority rule. In this section, we examine the NEHT-RB heuristic, modified by using different priority rules. In order to construct the priority rules examined in this work, the following measures need to be defined:

The average processing time of job j is defined as Eq. (3):

$$\mathrm{Avg}_j = \frac{1}{m}\sum_{i=1}^{m} p_{ij} \tag{3}$$

The standard deviation of processing times of job j is defined as Eq. (4):

$$\mathrm{Std}_j = \left[\frac{1}{m-1}\sum_{i=1}^{m}(p_{ij} - \mathrm{Avg}_j)^2\right]^{1/2} \tag{4}$$

Obviously, $m > 1$ is required here.

The average setup time of job j, where the setup times in all cases are considered, is defined as Eq. (5):

$$\mathrm{ST}_j = \frac{1}{2(n-1)m}\sum_{k=1,k\neq j}^{n}\sum_{r=1}^{m}(s_{rjk} + s_{rkj}) \tag{5}$$

Then the priority rules tested are as follows:

1. AvgStd
   This rule consists in sorting the jobs by decreasing $\mathrm{Avg}_j + \mathrm{Std}_j$. It implies that the larger average processing time (standard deviation of processing times) a job has, the higher priority it should have;

2. AvgStd+ST
   This rule consists in sorting the jobs by decreasing $\mathrm{Avg}_j + \mathrm{Std}_j + \mathrm{ST}_j$. When compared with the AvgStd rule, it implies that the larger average setup time a job has, the higher priority it should have. This is based on such a hypothesis: if a job with larger average setup time were scheduled at a later stage, it would disturb the partial schedule more strongly, and this would not help find a better solution;

3. AvgStd−ST
   This priority rule consists in sorting the jobs by decreasing $\mathrm{Avg}_j + \mathrm{Std}_j - \mathrm{ST}_j$. When compared with the AvgStd priority rule, it implies that the larger average setup time a job has, the lower priority it should have. This is based on such a hypothesis: if a job with larger average setup time were scheduled at a later stage, it would have more choice to find a good position, and this would help find a better solution.

The second and the third priority rules are in conflict on the usage of the average setup time in a certain sense. The aim of including these two rules is to examine how the average setup time can affect the performance. The original NEHT-RB heuristic modified by using these three priority rules can form three different heuristics. The experimental results for these heuristics in terms of average RPD on

SDST10, SDST50, SDST100 and SDST125 are listed in Table 3, where the RPDs are computed similar to Eq. (2).

Table 3. RPD comparison among the three priority rules on benchmarks

	AvgStd	AvgStd+ST	AvgStd−ST
SDST10	-0.17	-0.20	-0.14
SDST50	-0.17	-0.13	-0.27
SDST100	-0.11	-0.34	-0.26
SDST125	0.06	0.10	0.00

From this table, it can be seen that the average performance of these three heuristics is better than that of the original NEHT-RB heuristic on the SDST10, SDST50 and SDST100 benchmarks, while they are slightly worse than or equal to that of the NEHT-RB heuristic on the SDST125 benchmarks. Further, there is no one heuristic performing better than the others on all the benchmarks. Note that the priority rules used in the second and the third heuristics are in conflict in a certain sense, and both heuristics perform better than the first one on average. This shows that the setup times have an obvious effect on the performance of the heuristics. However, it is not very clear yet how to use the setup times to construct a much more efficient heuristic.

4 Tie-breaking Strategy

In the NEH heuristic [6], there may be ties when inserting a job into a partial schedule when solving the regular flowshop problem, i.e., several generated partial schedules having the same minimum makespan. In order to break these ties and find a more appropriate position for the inserting job, a tie-breaking strategy was proposed [3], the idea behind which is to choose the place more likely to balance the utilization of each machine. In the NEHT-RB heuristic for solving the SDST flowshop problem, the same problem arises. In this section, we extend the tie-breaking strategy proposed in our former work [3] by considering setup times and apply the modified strategy into the SDST flowshop problem to examine whether it works.

In order to choose a position by using the strategy, it should be computed the earliest possible completion time and the latest possible start time on machine i for job $\pi(k)$. The earliest possible completion time $C_{i,\pi(k)}$ can be computed by Eq. (1), while the latest possible start time $S_{i,\pi(k)}$ can be computed by Eq. (6):

$$S_{i,\pi(k)} = min\{S_{i+1,\pi(k)}, S_{i,\pi(k+1)} - s_{i\pi(k)\pi(k+1)}\} - p_{i\pi(k)} \quad (6)$$

where $S_{m,\pi(n)} = C_{m,\pi(n)} - p_{m\pi(n)}$, $S_{i,\pi(n)} = S_{i+1,\pi(n)} - p_{i,\pi(n)}$ for $i = 1, \ldots, m-1$ and $S_{m+1,\pi(k)}$ is a very large number for $k = 2, \ldots, n-1$.

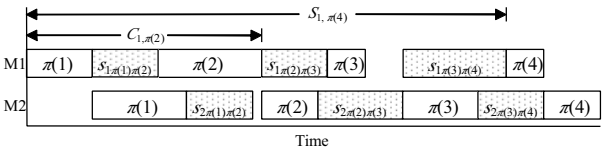

Figure 1. An example for $C_{i,\pi(k)}$ and $S_{i,\pi(k)}$

Then a partial schedule needs to be scheduled in this way: the jobs prior to the insertion place (including the inserting job itself if it is not the last job in the partial sequence) are scheduled as early as possible and the jobs succeeding the insertion place are scheduled as late as possible. Then the following measures can be computed for job $\pi(k)$:

$$E_{\pi(k)} = \frac{1}{m} \sum_{i=1}^{m} \frac{p_{i,\pi(k)}}{G_{i\pi(k)}} \quad (7)$$

$$D_{\pi(k)} = \sum_{i=1}^{m} \left[\frac{p_{i,\pi(k)}}{G_{i\pi(k)}} - E_{\pi(k)} \right]^2 \quad (8)$$

where $G_{i\pi(k)} = S_{i,\pi(k+1)} - s_{i\pi(k)\pi(k+1)} - C_{i,\pi(k-1)} - s_{i\pi(k-1)\pi(k)}$ indicates the gap for scheduling the ith operation of job $\pi(k)$.

Note that there exist exceptions: let $S_{i,\pi(k+1)}$ be equal to the latest possible completion time of job $\pi(k)$ on machine i if $\pi(k)$ is the last job in the partial sequence, and let $s_{i\pi(k)\pi(k+1)}$ be zero in this case; let $C_{i,\pi(0)}$ be equal to the earliest possible start time of job $\pi(1)$ on machine i, and let s_{i01} be zero; if $G_{i\pi(k)}$ is zero, then $p_{i,\pi(k)}$ must be zero, and so let $p_{i,\pi(x)}/G_{i\pi(k)}$ be zero in this case.

Then the choice of the insertion place in the third step of the original NEHT-RB [8] can be changed as: when inserting a job into a partial sequence, the place minimizing the makespan is firstly chosen; if there exist ties, the place k minimizing $D_{\pi(k)}$ is chosen; and if there still exist ties, then any of them is acceptable. Since the $C_{i,\pi(k)}$ and the $S_{i,\pi(k)}$ for all possible k can be computed in $O(nm)$ by using the speed-up method of Taillard [16], the modified NEHT-RB heuristic can be implemented in $O(mn^2)$.

Fig. 1 shows an example. In this example, we suppose the inserting job is $\pi(3)$, and then the $C_{1,\pi(2)}$ and the $S_{1,\pi(4)}$ are shown. The $C_{2,\pi(2)}$ and the $S_{2,\pi(4)}$ are similar to the above. Then the $E_{\pi(3)}$ and the $D_{\pi(3)}$ can be computed according to Eq. (7) and Eq. (8).

Then we apply this tie-breaking strategy into the three heuristics tested in Section 3 and the original NEHT-RB heuristic to study its influence. In Table 4, average RPDs for the four groups are reported, where the RPDs are computed similar to Eq. (2), and the Avg column indicates the results of the original NEHT-RB heuristic only modified by using the tie-breaking strategy.

Table 4. RPD comparison for the tie-breaking strategy

	Avg	AvgStd	AvgStd+ST	AvgStd−ST
SDST10	-0.02	-0.11	-0.20	-0.19
SDST50	0.01	-0.13	-0.12	-0.21
SDST100	-0.04	-0.21	-0.25	-0.15
SDST125	0.13	-0.01	0.06	-0.01

From this table, it can be seen that the tie-breaking strategy generally does not work on the original NEHT-RB heuristic, e.g., the average RPDs on SDST10 and SDST100 benchmarks are only -0.02 and -0.04 respectively, and those on SDST50 and SDST125 benchmarks are even 0.01 and 0.13 respectively. When comparing the results with those in Section 3, it can also be found that the tie-breaking strategy does not work on the three modified heuristics. This may be for that the ties are relatively rare, and the setup times for a certain job are different for different positions. All these show that the SDST flowshop problem is harder to solve than the regular flowshop problem.

5 Conclusion

In this work, three priority rules for the original NEHT-RB heuristic [8] are studied. The results show that these rules are helpful, especially on the instances with small setup times. When the setup times are averagely larger than the processing times, these rules fail. In this case, the PB heuristic by Tseng et al [18] performs better. According to Gupta and Darrow [5], setup times of many real-world instances are narrow, and so it is meaningful to improve the performance of the NEHT-RB by using the proposed priority rules. The results also show that the setup times have an obvious effect on the heuristics, but how to take advantage of them more efficiently needs to be studied further.

A tie-breaking strategy is also examined in this work. The strategy is extended from our previous work [3]. However, for the original NEHT-RB heuristic and its three variants, the extended strategy generally does not work. This also shows the hardness of solving the discussed problem.

Acknowledgment

This work is supported by Beijing Pivot Discipline Construction Project for Computer Application Technologies at Beijing Jiaotong University (XK100040519) and Special Foundation for Young Scientists of Beijing Jiaotong University (2008RC029).

References

[1] A. Allahverdi, C. Ng, T. Cheng, and M. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032, 2008.

[2] S. Das, J. Gupta, and B. Khumawala. A savings index heuristic algorithm for flowshop scheduling with sequence-dependent setup times. *Journal of the Operational Research Society*, 46(11):1365–1373, 1995.

[3] X. Dong, H. Huang, and P. Chen. An improved neh-based heuristic for the permutation flowshop problem. *Computers and Operations Research*, 35:3962–3968, 2008.

[4] J. Gupta. Flowshop schedules with sequence dependent setup times. *Journal of the Operations Research Society of Japan*, 29(3):206–219, 1986.

[5] J. Gupta and W. Darrow. The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24:439–446, 1986.

[6] M. Nawaz, J. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA*, 11(1):91–95, 1983.

[7] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice Hall, 2nd edition, 2001.

[8] R. Ríos-Mercado and J. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110:76–98, 1998.

[9] R. Ríos-Mercado and J. Bard. An enhanced tsp-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5:53–70, 1999.

[10] R. Ríos-Mercado and J. Bard. The flow shop scheduling polyhedron with setup times. *Journal of Combinatiorial Optimization*, 7(3):291–318, 2003.

[11] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165:479–494, 2005.

[12] R. Ruiz, C. Maroto, and J. Alcaraz. Solving the flowshop scheduling problem with sequence dependent setup times using advanced methaheuristics. *European Journal of Operational Research*, 165:34–54, 2005.

[13] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187:1143–1159, 2008.

[14] J. Simons. Heuristics in flow shop scheduling with sequence dependent setup times. *OMEGA*, 20(2):215–225, 1992.

[15] J. Stafford and F. Tseng. Two models for a family of flowshop sequencing problems. *European Journal of Operational Research*, 142:282–293, 2002.

[16] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[17] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[18] F. Tseng, J. Gupta, and J. Stafford. A penalty-based heuristic algorithm for the permutation flowshop scheduling problem with sequence-dependent set-up times. *Journal of the Operational Research Society*, 57:541–551, 2006.