Management and Production Engineering Review
Volume 13 • Number 1 • March 2022 • pp. 32–43
DOI: 10.24425/mper.2022.140874

Input Sequence of Jobs on NEH Algorithm for Permutation Flowshop Scheduling Problem

Radosław PUKA, Jerzy DUDA, Adam STAWOWY

AGH University of Science and Technology, Faculty of Management, Poland

Received: 18 November 2020
Accepted: 08 February 2022

Abstract
One of the most popular heuristics used to solve the permutation flowshop scheduling problem (PFSP) is the NEH algorithm. The reasons for the NEH popularity are its simplicity, short calculation time, and good-quality approximations of the optimal solution for a wide range of PFSP instances. Since its development, many works have been published analysing various aspects of its performance and proposing its improvements. The NEH algorithm includes, however, one unspecified and unexamined feature that is related to the order of jobs with equal values of total processing time in an initial sequence. We examined this NEH aspect using all instances from Taillard's and VRF benchmark sets. As presented in this paper, the sorting operation has a significant impact on the results obtained by the NEH algorithm. The reason for this is primarily the input sequence of jobs, but also the sorting algorithm itself. Following this observation, we have proposed two modifications of the original NEH algorithm dealing with sequencing of jobs with equal total processing time. Unfortunately, the simple procedures used did not always give better results than the classical NEH algorithm, which means that the problem of sequencing jobs with equal total processing time needs a smart approach and this is one of the promising directions for further research.

Keywords
Job and activity scheduling, Scheduling, Optimization, Permutation flowshop scheduling problem, NEH algorithm, Input sequence.

Introduction

This paper presents an analysis of an unexamined feature of the NEH algorithm applied for the permutation flowshop scheduling problem, in which n jobs are to be processed consecutively on m machines and the order of jobs on every machine is to be kept the same. The solution to the problem is a permutation containing all n jobs, and the completion time of the last job on the last machine (makespan) should be minimized.

NEH algorithm, the most popular heuristic for PFSP proposed by Nawaz, Enscore and Ham (1983), is described in the following steps:

1. Sorting jobs (from an input sequence) to generate an initial sequence based on the sums of their processing times arranged in a non-increasing order.
2. Scheduling the first two jobs in the sequence in such a manner that the makespan for these jobs should be the smallest.
3. The next job k $(2 < k < n)$ that has not been sequenced yet is inserted in the existing sequence into a position that minimises the makespan for these jobs. The insertion is repeated until all jobs are sequenced.

The NEH algorithm has two properties unresolved in the original publication:

1. Equal total processing time (TPT) of some jobs (step 1).
2. Equal makespan of partial sequences (step 2 and step 3).

While the second listed aspect has been widely studied and various tie-breaking techniques have been proposed – see e.g. Nagano and Moccelin (2002) and Liu et al. (2017) – the first one has been noticed, but only partially examined. At most, in the case of equal total processing time of some jobs, a random order of these jobs in an initial sequence is proposed (e.g. Ying and Lin, 2013), without any justification. Thus, the NEH constructive algorithm, which by definition should give repetitive results, becomes a partially random algorithm that generates the more different solu-

***Corresponding author**: Adam Stawowy – AGH University of Science and Technology, Faculty of Management, 30-067 Kraków, ul. Gramatyka 10, phone: +48 12 617 46 76, e-mail: astawowy@zarz.agh.edu.pl*

tions, the more jobs with an equal sorting criterion occur in the input sequence. Although researchers have proposed various ways to generate the initial order of jobs, the purpose of those activities is not to resolve the ambiguity identified, but simply to create a sorting criterion that will yield better results than the original approach.

The most comprehensive analysis of this problem is contained in Vasiljevic and Danilovic (2015). The authors emphasize that in contrast to the sort of ties in the insertion phase, the ties in the initial phase are not defined in the definition of NEH. This results in inaccuracy that can be observed in a large number of experimental results published on this topic. Vasiljevic and Danilovic provided the experiments that confirm the importance of the information about the sort of ties in the initial phase in any experimental result related to NEH. The conclusion, obtained by their study, is that the range of the objective values for different sorting approaches of ties is often greater than the NEH improvements, proposed in the literature.

In order to examine in details the input sequence problem in the NEH algorithm, the benchmark set proposed by Taillard (1993) and Vallada-Ruiz-Framinan (VRF) benchmark proposed by Vallada et al. (2015) are used further in the paper. The basic statistics for test instances are given and some ideas dealing with sequencing problem are proposed. The aim of our experiments is to show that ambiguous sorting of jobs with the same total processing time has an impact on the results of the NEH algorithm and should be taken into account while developing its extensions. We considered only the original version of NEH, in which the sorting criterion is a total processing time of jobs on all machines. As it has been shown in (Framinan et al., 2003), it is the best single criterion for NEH heuristic dealing with makespan minimization.

The paper has the following structure. Background presents the literature review of the NEH heuristic and its extensions. The explanation of the job sequencing problem is described in the Third Section. The Fourth Section gives the detailed analysis of the test instances. The computational experiments are summarized in the Fifth Section, and two approaches solving input sequence problem are proposed in Section Potential directions. The conclusions are drawn in the last Section.

Background

There are many studies reported in the literature devoted to the permutation flowshop scheduling problem. a comprehensive review on PFSP formulations as well as solution approaches are presented in (Rossi et al., 2016) and (Fernandez-Viagas et al., 2017). For practical reasons, this Section gives a limited review of the methods used to solve the considered problem: we focus only on approaches that are closely related to our work, i.e. NEH properties and NEH improvements.

Due to its simplicity and the quality of the results generated, the NEH algorithm still remains the most popular construction heuristics for the flowshop scheduling problem and forms the basis for further research on algorithms solving this problem. NEH has a complexity of $O(m \cdot n^3)$, but Taillard (1990) has shown that its complexity can be reduced to $O(m \cdot n^2)$. Ruiz and Maroto (2005) have examined 25 different heuristics and have shown that the NEH algorithm gives the best results among all heuristics examined and, additionally, in a much shorter time.

Dong, Huang and Chen (2008) presented a variant of the NEH heuristic based on the deviation of processing times (NEH-D). It employs a special priority rule for the sorting phase of the NEH algorithm assigning higher priority to the jobs that have a larger deviation of the processing times on each machine. The authors used the sum of average processing time and the standard deviation of processing times for the first phase of their NEH-D algorithm. In the second phase (inserting jobs into a partial sequence) they developed a special mechanism able to decide which job should be taken if the partial sequence for them is equal (a tie situation). With the computational complexity of $O(m \cdot n^2)$, when using Taillard speed-up method, NEH-D is able to provide better solutions than the original NEH algorithm by 11.9% (Rossi et al., 2016) in terms of ARPD (Average Percentage Relative Deviation). At the same time Kalczynski and Kamburowski (2007) proposed another tie-breaking strategy in the second phase of the NEH algorithm that is based on makespan calculation for partial sequences. However, an average gain over the original NEH algorithm was only 5.1%. Later in (2008) and (2009) Kalczynski and Kamburowski proposed new strategies for both phases of the NEH algorithm, but the results on average were not better than 7.9%, comparing to the original NEH algorithm. Rad, Ruiz and Boroojerdian (2009) proposed a more complex heuristic called FRB3 in which all jobs from the current sequence were reinserted in the second phase of the NEH method. Such strategy led to an average gain of 48.0% over the NEH algorithm, however, for the cost of the increased computational complexity of $O(m \cdot n^4)$. Finally, Fernandez-Viagas and Framinan (2014) used a tie-breaking strategy that selects

the sequence minimizing the estimation of total idle time. Such strategy gave better results than the original NEH algorithm by 8.2% on average with the complexity of $O(m \cdot n^2)$. The most popular extensions of the original NEH algorithm have been recently compared by Rossi, Nagano and Nero (2016). The authors also present yet another approach that can bring an average improvement of 57.6% over the original NEH heuristic, however average computational time was twice as big as in the case of FRB3 heuristic, so far the most demanding in this respect.

According to Vasiljevic and Danilovic (2016), many NEH improvements are described in a very imprecise manner, which makes it impossible to determine whether the advantages (simplicity, efficiency, low computational complexity) of the original algorithm are met.

The review presented above shows that there is still a room for improvement of the original NEH algorithm, however, more studies analysing the impact of job processing times patterns are required. Improvements to the NEH algorithm apply to either the criteria for building an initial order or a tie-breaking procedure.

The problem of input sequence of jobs in NEH algorithm

Various results of NEH performance for the same instances of known benchmarks have been reported in the literature (e.g. Semančo and Modrák, 2012 vs Ying and Lin, 2013). Such an ambiguous operation of the well-known, constructive algorithm results from different interpretations of sorting (step 1) and selection of the best partial sequence (next steps). In the case of the sorting step three issues should be considered:

1. What order is accepted for jobs with equal total processing times?
2. Is it possible that different sorting algorithms give different results?
3. Can the input sequence of jobs cause NEH to give different results?

Let T_{Ji} and T_{Jj} be the total processing time of job J_i and job J_j ($i <> j$), respectively. Sorting in a "non-increasing order" is trivial if $T_{Ji} > T_{Jj}$ or $T_{Ji} < T_{Jj}$: in the first case job J_i will precede job J_j in the initial sequence, in the second case – the sequence will be reversed. But if $T_{Ji} = T_{Jj}$, the initial sequence will depend on input (before sorting) order of jobs, on the inequality operator used ("$>$" vs "$\geq$"), and – what will be shown later – the sorting algorithm itself.

To illustrate the above mentioned operating principle of the NEH algorithm we have used the processing times contained in Table 1 and comprising 5 jobs processed on 3 machines. We do not want to draw attention to the problem of tie-breaking in the jobs insertion step, but to different results of the NEH algorithm depending on the input order of jobs with equal total processing time (in the described case these are jobs no. 3 and 4).

Table 1
Sample input data for PFSP

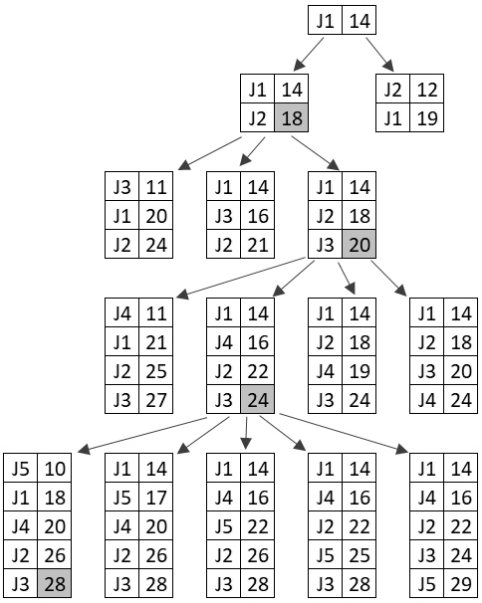
Job/mach.	M_1	M_2	M_3	TPT
J_1	3	7	4	14
J_2	5	3	4	12
J_3	5	4	2	11
J_4	5	5	1	11
J_5	3	4	3	10

The problem is presented in three scenarios; Figures 1 and 2 illustrate the results: the Figures contain indexes of jobs as well as the completion time of each job. In the first scenario, the criterion for sorting jobs is given by strict inequality $T_{Ji} > T_{Jj}$ and the order of jobs with equal processing time in the input sequence is $J_3 - J_4$ (Figure 1a). In the second scenario, the sorting criterion is the same, but the input order of jobs with equal processing time is $J_4 - J_3$ (Figure 1b).

It is clear that in these two scenarios the first job from a subsequence of jobs with equal times is always the first in the initial sequence (after sorting). In other words, the subsequence of jobs with an equal indicator in the initial sequence will always be the same as the order of those jobs in the input sequence. In the third scenario, the criterion of sorting jobs is given by weak inequality $T_{Ji} \geq T_{Jj}$ (Figure 2). Also, in this case, tie-breaking in the second step was not considered, and the difference in the ranking results only from weakening inequalities in the insertion phase ("$\geq$" instead of "$>$"). If weak inequality is used the subsequence of jobs with equal total processing time in the initial sequence will always be the reversed order of those jobs in the input sequence.

Based on the presented example, it can be concluded that both the strictness of the inequality in the sorting process and the input order of jobs have an influence on the final value of the makespan. Moreover, we can say that the original NEH is, in fact, a non-deterministic algorithm, because the resulting solutions depend on the *de facto* accidental input order of jobs (if at least two jobs exist with equal total processing time).

a)



b)

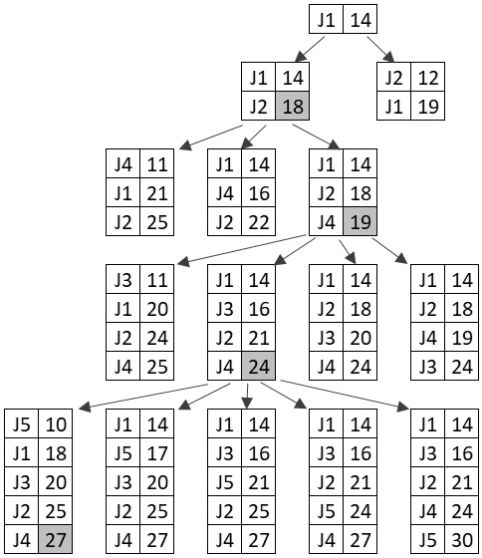


Fig. 1. Different results of the NEH algorithm depending on the sequence of jobs with equal total processing time

There is another problem with the first step of the NEH algorithm: the silent assumption is that the initial sequence of jobs is independent of the sorting algorithm used, but our research indicates that this is not true. We have compared the NEH results for two commonly used sorting algorithms, i.e. bubble sort (BS) and Quicksort (QS): the NEH results over Taillard and VRF benchmark sets are summarized in Table 2. As a performance measure (and further in the text) the ARPD (Average Relative Percent Deviation) was used. The ARPD is calculated by $(C\max_A - Best)/Best \times 100\%$, where $C\max_A$ is

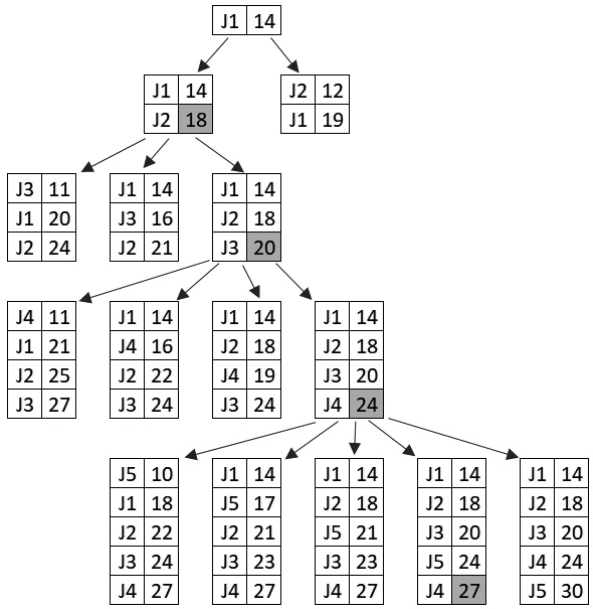


Fig. 2. The result of the NEH algorithm for non-strict inequality of sorting criterion

makespan obtained by the algorithm under consideration, *Best* is the corresponding best-known makespan found in the literature.

Table 2
Summary of NEH results for different sorting algorithms employed in step 1

Benchmark/ sorting algorithm	Taillard		VRF small		VRF large*	
	QS	*BS*	*QS*	*BS*	*QS*	*BS*
Average ARPD from all instances [%]	3.35	3.32	3.87	3.84	3.32	3.33
Average ARPD for instances where all jobs have different TPT [%]	4.21	4.21	2.91	2.91	–	–
Average ARPD for instances with at least two jobs with equal TPT [%]	3.18	3.15	4.30	4.26	3.32	3.33
Number of instances with better results than a competitive sorting algorithm (from instances with at least two jobs with equal TPT)	37/ 100	40/ 100	45/ 166	33/ 166	116/ 240	122/ 240

* all instances of VRF large contain at least two jobs with equal TPT

Differences only apply to the instances with jobs with equal TPT and result from the way the Quicksort algorithm works (QS is an unstable algorithm which means that it does not preserve the original order of elements when sorting by a non-unique value).

Although the differences between results for the NEH algorithm employing the two sorting algorithms examined are not big, they are unacceptable in the case of a constructive algorithm. They indicate also that the sorting algorithm is important for determining the initial sequence and only stable sorting algorithms (e.g. bubble, block, insertion, merge, odd-even sort) should be used. This is an unexpected conclusion pointing to one more ambiguity (apart from the previously discussed) of the first stage of the NEH algorithm. However, this aspect is also directly related to the problem of equal TPT for some jobs.

Analysis of test instances

As the most common benchmarks for the permutation flowshop scheduling problem are 120 instances presented by Taillard and 480 instances of VRF, the statistical analysis has been performed on these benchmarks. We report the average number of subsequences with k ($k = 2, 3, 4$; $k \geq 5$) jobs with equal TPT, and related to the number of these subsequences: average, minimum and maximum number of possible NEH initial sequences. For example, if we have two subsequences with jobs with equal TPT in the input sequence, one containing 2 jobs and another containing 4 jobs, we can generate 48 ($2 \cdot 4$) different initial sequences. The results are presented in Tables 3, 4 and 5 for Taillard, small VRF and large VRF

Table 3
Basic statistics for Taillard instances

$n \times m$	The average number of subsequences with k jobs				The number of possible sequences of NEH initial sequence		
	$k = 2$	$k = 3$	$k = 4$	$k \geq 5$	Avg.	Min	Max
205	0.8	0.0	0.0	0.0	2.3	1	8
20×10	0.3	0.1	0.0	0.0	1.9	1	6
20×20	0.5	0.0	0.0	0.0	1.6	1	4
50×5	5.0	0.4	0.0	0.0	304.0	16	2304
50×10	3.0	0.1	0.0	0.0	14.4	4	48
50×20	2.1	0.0	0.0	0.0	10.6	1	64
100×5	13.0	1.7	0.7	0.0	1.3E9	5.1E2	1.2E10
100×10	10.0	1.7	0.0	0.0	7.3E5	3.8E2	7.1E6
100×20	7.6	0.4	0.0	0.0	2.0E3	1.6E1	1.2E4
200×10	30.0	6.5	1.0	0.2	2.6E17	2.2E13	2.3E18
200×20	23.6	3.7	1.0	0.1	4.1E14	1.5E8	4.0E15
500×20	80.7	32.4	10.3	2.3	1.5E75	1.5E61	1.5E76

Table 4
Basic statistics for VRF small instances

$n \times m$	The average number of subsequences with k jobs				The number of possible sequences of NEH initial sequence		
	$k = 2$	$k = 3$	$k = 4$	$k \geq 5$	Avg.	Min	Max
$10 \times m$	0.0	0.0	0.0	0.0	1.0	1	2
$20 \times m$	0.5	0.0	0.0	0.0	1.6	1	6
$30 \times m$	1.3	0.1	0.0	0.0	5.2	1	48
$40 \times m$	1.7	0.2	0.0	0.0	6.7	1	48
$50 \times m$	3.2	0.2	0.0	0.0	29.8	1	384
$60 \times m$	4.1	0.3	0.1	0.0	140.3	1	1152

Table 5
Basic statistics for VRF large instances

$n \times m$	The average number of subsequences with k jobs				The number of possible sequences of NEH initial sequence		
	$k = 2$	$k = 3$	$k = 4$	$k \geq 5$	Avg.	Min	Max
$100 \times m$	6.9	0.6	0.0	0.0	6.3E3	3.2E1	1.5E5
$200 \times m$	2.6	3.0	0.3	0.0	4.3E13	6.6E4	1.2E15
$300 \times m$	40.8	8.2	1.3	0.1	1.6E27	2.4E14	2.5E28
$400 \times m$	62.2	14.7	3.2	0.7	1.0E51	2.5E24	3.2E52
$500 \times m$	82.0	22.5	6.5	1.3	7.7E75	7.7E36	2.5E77
$600 \times m$	95.8	34.8	9.6	3.4	1.4E103	9.1E56	4.4E104
$700 \times m$	110.0	43.6	15.3	6.8	1.5E140	1.2E74	3.6E141
$800 \times m$	123.0	53.0	22.0	10.8	8.8E174	8.9E101	2.6E176

instances, respectively. As the VRF benchmarks are very extensive, the statistics for them are given only for the number of jobs n (without breaking down into the number of machines m).

On the basis of the presented statistics we have observed the following:

- 83.3% of the Taillard instances (100 out of 120) and 84.6% of the VRF instances (406 out of 480) contain at least two jobs with equal total processing time,
- the number of subsequences with k ($k = 2, 3, 4$; $k \geq 5$) jobs in an input sequence and the related number of possible initial sequences increases exponentially with the increase in the number of jobs n,
- the smaller the number of machines with the same number of jobs, the more jobs with the same TPT exists,
- all instances from VRF large set and Taillard large set ($n \geq 100$ jobs) contain at least 2 jobs with equal TPT.

The collected statistics indicate the importance of the problem of equal total processing times for the stable operation of the NEH algorithm and for possible attempts to improve it.

Computational experiments

The aim of the experiments was to assess the influence of the presence of jobs with equal TPT in the input sequences on the results obtained by the classical NEH algorithm and its variants. For computational reasons we decided to perform an exhaustive search (ES) for small-size instances (VRF small and Taillard $n < 100$) and random search (RS) for large-sized instances (VRF large and Taillard $n \geq 100$). In all small cases, the maximum possible sequences (denoted as Max in Tables 3 and 4) were generated and evaluated while for large instances 1000 and 5000 random sequences were evaluated. "Random" in this case means a random order of jobs in subsequences containing jobs of the same TPT – remaining jobs are taken according to their order in the initial sequence. In all cases the Taillard acceleration procedure was used.

For the implementation of the algorithm, Python 2.7 was used as the programming language while the code was run in the PyPy environment to speed up the calculations. Computations were performed on a computer with Xeon 1220 CPU 3.8 GHz.

Results for NEH

The results obtained for ES and RS are presented in Tables 6–9 and they are compared to the results achieved by the classical version of the NEH algorithm. The tables contain mean, worst and best of ARPD for those and only those instances in which TPT problems are present. As VRF instances with 10 jobs contain only one interesting instance and with only one pair of jobs with equal TPT, this case – as statistically irrelevant – was not included in Table 8.

Based on the above data we investigated the confirmatory hypothesis that the proper sequence of TPT jobs has a significant effect on NEH results. a Wilcoxon signed-rank test was used to compare differences in outcomes between NEH and ES/RS. Confirmatory testing at a significance level of 5% showed statistically significant improvements in all instances ($p < 0.001$). We can also observe that there are significant differences between the results obtained by the random search with different number of evaluated sequences: RS1000 and RS5000.

Table 6
ARPD [%] of exhaustive search for Taillard small instances ($n < 100$)

$n \times m$	NEH	ES		
		Worst	*Best*	*Mean*
20×5	3.0214	3.4687	2.6699	3.0693
20×10	4.2063	5.6178	4.2063	4.9121
20×20	3.9564	4.1499	3.1245	3.9749
50×5	0.7272	1.1917	0.3720	0.6850
50×10	5.0729	5.7989	4.6917	5.1581
50×20	6.8798	7.1504	6.0551	6.5637
Average	3.9773	4.5629	3.6325	4.0605

Table 7
ARPD [%] of random search for Taillard large instances ($n \geq 100$)

$n \times m$	NEH	RS1000			RS5000		
		Worst	*Best*	*Mean*	*Worst*	*Best*	*Mean*
100×5	0.53	0.99	0.21	0.49	1.02	0.19	0.50
100×10	2.22	3.63	1.17	2.26	3.74	1.11	2.26
100×20	5.35	6.83	4.45	5.57	6.88	4.45	5.57
200×10	1.26	2.22	0.75	1.33	2.32	0.64	1.33
200×20	4.41	5.71	3.18	4.40	5.90	3.07	4.39
500×20	2.07	2.86	1.56	2.17	2.95	1.49	2.16
Average	2.64	3.71	1.89	2.70	3.80	1.82	2.70

Table 8
ARPD [%] of exhaustive search for VRF small instances

$n \times m$	NEH	ES		
		Worst	*Best*	*Mean*
$20 \times m$	3.7486	3.7922	3.4555	3.6153
$30 \times m$	4.4966	4.8497	4.1738	4.4967
$40 \times m$	4.2235	4.5939	3.8818	4.2261
$50 \times m$	4.3576	4.9590	3.8606	4.3834
$60 \times m$	4.2258	5.2415	3.5990	4.3369
Average	4.2104	4.6873	3.7941	4.2117

Results for other NEH-based algorithms

In order to check the significance of the problem of sequencing jobs with equal total processing time, the performance of the NEHKK algorithm (Kalczynski and Kamburowski, 2007) was analysed. NEHKK uses the same initial sorting as the NEH algorithm (sum of job processing times in a non-increasing order). The NEHKK algorithm additionally employs a tie-breaking approach in the inserting phase. The results are presented in Table 10 and Table 11 for the same benchmarks as were considered in previous Section. ES, RS1000, and RS5000 means the best ARPD results.

Table 9
ARPD [%] of random search for VRF large instances

$n \times m$	NEH	RS1000			RS5000		
		Worst	*Best*	*Mean*	*Worst*	*Best*	*Mean*
$100 \times m$	5.37	6.36	4.42	5.34	6.38	4.40	5.34
$200 \times m$	4.45	5.65	3.37	4.46	5.84	3.20	4.46
$300 \times m$	3.67	4.59	2.79	3.66	4.78	2.67	3.66
$400 \times m$	3.26	4.01	2.43	3.20	4.11	2.37	3.20
$500 \times m$	2.86	3.55	2.17	2.83	3.64	2.10	2.83
$600 \times m$	2.55	3.19	1.97	2.55	3.26	1.90	2.55
$700 \times m$	2.31	2.89	1.81	2.31	2.95	1.71	2.31
$800 \times m$	2.13	2.70	1.70	2.18	2.77	1.63	2.18
Average	3.33	4.12	2.58	3.32	4.22	2.50	3.32

Table 10
ARPD [%] of NEHKK and exhaustive search for small benchmarks

Instances	NEH	NEHKK	ES
Small Taillard	3.9773	3.8148	3.3960
Small VRF	4.2104	4.0387	3.5891

Table 11
ARPD [%] of NEHKK and random search for large benchmarks

Instances	NEH	NEHKK	RS1000	RS5000
Large Taillard	2.6383	2.5284	1.7742	1.7207
Large VRF	3.3316	2.6831	2.0391	1.9617

Similarly, as in the case of the NEH algorithm, a Wilcoxon signed-rank test confirms that sequencing jobs with equal TPT has a significant impact on the NEHKK results.

In order to emphasize the importance of the problem of jobs with equal total processing time, the results of the NEH and NEHKK algorithms were compared with the results of other algorithms presented in the review paper (Fernandez-Viagas et al., 2017). The summary shown in Table 12 presents the results for

the large Taillard instances used in the previous tests. The results obtained (quoted only to outline the problem, not to compare the efficiency of the algorithms) are sorted by decreasing ARPD values.

Table 12
ARPD [%] of NEH-based algorithms for large Taillard benchmark

Algorithm*	ARPD	Algorithm*	ARPD
RAER	3.0514	KKER-di	2.2810
RAER-di	2.7931	NEHD-di	2.1981
NEH	2.6383	FRB4_2	1.9721
NEHKK	2.5284	NEH RS1000	1.8860
NEMR	2.5080	NEH RS5000	1.8247
NEH-di	2.4851	FRB4_4	1.8131
NEHR	2.4801	NEHKK RS1000	1.7742
NEHKK2	2.4754	NEHKK RS5000	1.7207
NEHKK1-di	2.4737	FRB4_8	1.6668
NEH1-di	2.4549	FRB4_6	1.6152
KKER	2.4211	FRB2	1.6131
CL_WTS	2.3565	FRB4_12	1.5604
NEHR-di	2.3481	FRB4_10	1.5386
NEHFF	2.3146	FRB3	1.3459
NEMR-di	2.2999	FRB5	1.1403

* abbreviations – according to Fernandez-Viagas, Ruiz and Framinan (2017), page 709

Based on the data presented in Table 12 it can be concluded that the NEH algorithm could potentially give better results than most of its improvements presented in the literature. The results of the NEHKK algorithm are even more promising (see Table 12). It is worth noting again that both algorithms use the same initial order of jobs which are sorted in non-increasing order by TPT. We did not modify the operating principles of the analysed algorithms, but only examined what are the effects of the inaccuracy resulting from the lack of sorting rules.

Potential directions

Previous Section has proved that the NEH algorithm and its variants can be significantly improved by the proper ordering of jobs with equal TPT that are present in the input sequence. There are two obvious ways to address this problem:

1. Developing a sorting criterion that uniquely identifies the initial sequence.
2. Developing a procedure that gives the best order/s of jobs with equal TPT in sub-sequence/s.

Of course, the NEH algorithm extensions developed over the last few years, which use complex sorting criteria, somehow determine an unambiguous initial sequence. However, our research shows that such methods do not solve the considered problem – these algorithms usually give worse results than NEH RS5000 (see Table 12). For example, ARPD for NEH RS5000 is on average 35.1% better than for NEHKK1 that uses a non-increasing sum of weighted processing times as a priority rule and 20.1% than for NEH-D that uses a descending sum of mean and standard deviation of processing times. Therefore we decided to check two simple approaches for generating the proper order in subsequences of jobs with equal TPT.

Both of the proposed approaches use a greedy mechanism of inserting jobs with equal total processing times into the sequence, searching for locally best solutions. In both algorithms strict inequality was applied in the criterion of inserting jobs for scheduling and no tie-breaking rule was employed (same as in the original NEH algorithm – the first position is selected when ties exist).

NEH-First algorithm

The NEH-First algorithm is a modification of the NEH algorithm that uses the following three-steps approach to select jobs with equal total processing time:

1. For s scheduled jobs (in accordance with the NEH algorithm), a check is made of what the makespan will be after inserting each of the k jobs with the same total processing time into the sequence.
2. The job for which the completion time is the smallest is selected for insertion into the sequence. The number of jobs in the sequence increases to $s+1$, while the number of jobs k decreases by 1 (by the job inserted into the sequence).
3. If k is greater than 1, the algorithm returns to step 1; otherwise, execution of the algorithm is continued based on the operation scheme of the classical NEH algorithm until another occurrence of jobs with equal total processing time.

The operation of the NEH-First algorithm has been demonstrated in Figure 3, based on the data from Table 1. For the example presented, the modification distinguishing between the NEH-First algorithm and the NEH was only used once because only one pair of jobs (no. 3 and 4) had equal total processing times. In the NEH-First algorithm, the computations enable the selection of the job (from the set of jobs with equal processing times), the insertion of which allows to obtain

the shortest makespan for the given set of jobs. After inserting one job from the two-element set of equal processing time jobs, there is only one remaining job that should be added to the sequence and therefore the further operation of the algorithm was identical to the operation of the original NEH algorithm.

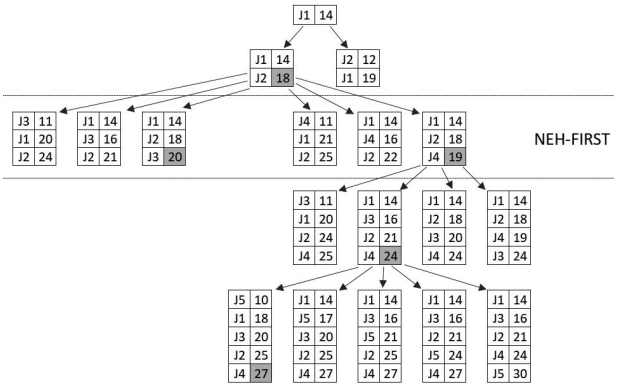


Fig. 3. Operation scheme of the NEH-First algorithm

The essence of the NEH-First algorithm is inserting such a job from the list of jobs with equal processing times for which the makesapn will be the smallest and then repeating this operation for the other, not yet sequenced jobs. This means that for one set of k jobs with equal total times, the NEH-First algorithm will be run by $\left(\sum_{i=1}^{k-1} i\right)$ times more (for the given sequence) than the original NEH algorithm.

NEH-Combination algorithm

Operation of NEH-Combination for equal TPT jobs can be described by the following steps:

1. For s scheduled jobs, all permutations of the possible sequences of inserting k jobs with equal total processing times are generated (e.g. for three jobs: A, B and C, the following permutations will be generated: ABC, ACB, BCA, BAC, CAB, CBA).
2. Jobs forming the generated permutations are inserted into the sequence using the NEH insertion procedure. The selected permutation is the one that allows obtaining the minimum makespan.

Operation of the NEH-Combination algorithm has been presented in Figure 4.

Performance of the NEH-Combination algorithm consists in checking all the possible permutations of the set of k jobs with equal total processing times. For each of the permutations, completion time is computed for the last job in the permutation in the way corresponding to the operation of the NEH algorithm. Checking the time for all possible permutations causes an increase in the computational complexity for each

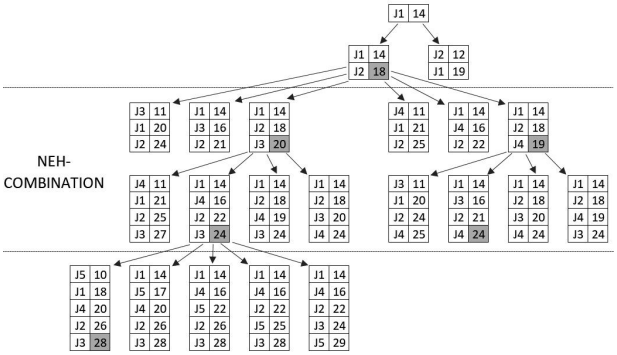


Fig. 4. Operation scheme of the NEH-COMBINATION algorithm

set of k jobs with equal processing times by $k!$-1 compared with the classical NEH algorithm.

Analysis of algorithms' results

For both algorithms, a limit of the maximum calculation time for a single instance has been set to 30 minutes. This time was exceeded by the NEH-Combination algorithm for four VRF benchmark instances (three instances in 700x20 set and one instance in 800x20 set). Therefore, it is impossible to present the average results obtained by the NEH-Combination heuristics for the large VRF benchmark. It should be noted that there are cases for which the NEH-Combination algorithm works inefficiently in terms of computational time. ARPD values for the NEH-First and NEH-Combination algorithms are presented in Tables 13 and 14. Table 15 presents average execution times for the instances containing a minimum of two jobs with equal total processing times for Taillard and VRF benchmarks.

From the above data, it can be concluded that the proposed approaches did not always allow to obtain better results than the NEH algorithm. It can, therefore, be concluded that both – quite advanced approaches – do not work as expected, because the proper input sequence must take into account the permutation of all jobs with equal TPT. In the case of the proposed algorithms, if there are several subsequences of jobs with equal TPT, these subsequences are considered consecutively, independently of each other. It appears that this local approach is not sufficient and that the overall (global) sequence of tasks in all subsequences should be taken into account. This conclusion is confirmed by the data in Tables 6–9 and Tables 13–14: in all cases the results of our algorithms are much worse than the best results for NEH ES and NEH RS.

From the results reported in Table 15, it can be seen that both approaches are more time-consuming than

Table 13
ARPD [%] of proposed approaches for Taillard instances

Small Taillard				Large Taillard			
$n \times m$	NEH	NEH-First	NEH-Comb	$n \times m$	NEH	NEH-First	NEH-Comb
20×5	3.02	3.04	3.12	100×5	0.53	0.46	0.54
20×10	4.21	4.21	5.35	100×10	2.22	2.22	2.19
20×20	3.96	3.99	3.80	100×20	5.34	5.50	5.52
50×5	0.73	0.58	0.75	200×10	1.26	1.31	1.25
50×10	5.07	5.05	5.07	200×20	4.41	4.41	4.18
50×20	6.88	6.60	6.66	500×20	2.07	2.07	2.02
Avg.	3.98	3.91	4.13	Avg.	2.64	2.66	2.62

Table 14
ARPD [%] of proposed approaches for VRF instances

Small VRF				Large VRF			
$n \times m$	NEH	NEH-First	NEH-Comb	$n \times m$	NEH	NEH-First	NEH-Comb
$20 \times m$	3.75	3.72	4.18	$100 \times m$	5.37	5.49	5.71
$30 \times m$	4.50	4.54	4.67	$200 \times m$	4.50	4.56	4.80
$40 \times m$	4.22	4.25	4.35	$300 \times m$	3.67	3.59	4.02
$50 \times m$	4.36	4.38	4.72	$400 \times m$	3.26	3.15	3.66
$60 \times m$	4.23	4.44	4.48	$500 \times m$	2.86	2.74	3.36
				$600 \times m$	2.55	2.47	3.03
				$700 \times m$	2.31	2.20	–
				$800 \times m$	2.13	2.08	–
Avg.	4.21	4.27	4.48	Avg.	3.33	3.28	–

Table 15
Average algorithms' execution times [sec]

Benchmark	NEH	NEH-First	NEH-Comb
Small Taillard	2.23	2.38	2.75
Large Taillard	78.73	110.20	439.05
Small VRF	2.80	2.97	3.03
Large VRF	745.56	1099.33	N/A

NEH, but the differences of computational time are relatively small, especially for small-size instances. On the other hand, the computational time of the NEH-Combination algorithm for the large VRF benchmark exceeded the accepted time limit. These differences are natural, as additional time is needed to calculate the variants of subsequences containing jobs with equal TPT.

It is an interesting question why much complicated NEH-Combination approach gives worse results than the First one (only results of large Taillard benchmark set are slightly better). One reason is that the First approach works on a principle more similar to the original NEH than the Comb approach, which makes better use of the NEH algorithm advantages. Another reason may be that a better partial schedule generated in the beginning stage of the Combination approach did not guarantee a better complete schedule that were finally obtained.

Proposition

The above analysis showed that there is no obvious way to solve the problem under consideration – despite the use of seemingly promising approaches, no better results than NEH were achieved. There is no doubt that a more complex, perhaps intelligent, way

of arranging the input sequence of jobs with equal TPT should be developed. a temporary solution to the problem may be as follows:

- for small instances (up to 60 jobs) ES can be used,
- for large instances (above 60 jobs) RS1000 can be suitable.

Both approaches provide a significant improvement in NEH results (see Tables 6–9), but at the cost of a significant increase in calculation time (several dozen times for ES and 1000 times for RS1000). Nevertheless, it can be a good reference point for designing algorithms to address the problem of equal total processing time of some jobs in the input sequence.

Conclusions

In this article, we pointed out two problems related to the performance of the well-known NEH algorithm. First of all, we showed that the order of jobs with the same total processing time in the initial sequence affects the results of the makespan calculations. Secondly, we demonstrated that sorting algorithms can give different initial sequences, which causes the NEH algorithm to work ambiguously. It means that the method of sorting jobs with equal total processing times should be included as a parameter of the NEH algorithm or should be included in the name of the algorithm (similarly to the introduced tie-breaking mechanism for the job insertion phase). This is related to the results obtained by us, which may potentially be much better (as in the case of NEH ES and NEH RS) from the results of the NEH itself.

We tried to find a method for sequencing jobs with equal total processing time, which would allow obtaining results even at least similar to the best results of NEH ES and NEH RS. The study analysed the possibility of using two local searching approaches to neutralize the effects of using different sorting methods for jobs with equal total processing time. The proposed NEH-First and NEH-Combination approaches did not always allow to obtain results better than the NEH algorithm not implementing such local searching.

In our opinion, the work aimed at determining the proper order of jobs in subsequences with the same TPT should be carried out in three directions:

- an unambiguous order in the input sequence (e.g. pre-sorting) to avoid the random operation of the NEH algorithm,
- a more complicated (smarter) way of generating the order of these subsequences,
- application of alternative indicators for TPT allowing the unambiguous resolution of the sorting problem.

Acknowledgments

This study was conducted under a research project funded by a statutory grant of the AGH University of Science and Technology in Kraków for maintaining research potential.

References

Dong X., Huang H. and Chen P. (2008), An improved NEH-based heuristic for the permutation flowshop problem, *Computers and Operations Research*, Vol. 35, No. 12, pp. 3962–3968. DOI: 10.1016/j.cor.2007.05.005

Fernandez-Viagas V. and Framinan J.M. (2014), On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem, *Computers and Operations Research*, Vol. 45, pp. 60–67. DOI: 10.1016/j.cor.2013.12.012

Fernandez-Viagas V., Ruiz R. and Framinan, J.M. (2017), a new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation, *European Journal of Operational Research*, Vol. 257, No. 3, pp. 707–721. DOI: 10.1016/j.ejor.2016.09.055

Framinan J.M., Leisten R. and Rajendran C. (2003), Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem, *International Journal of Production Research*, Vol. 41, No. 1, pp. 121–148. DOI: 10.1080/00207540210161650

Kalczynski P.J. and Kamburowski, J. (2007), On the NEH heuristic for minimizing the makespan in permutation flow shops, *Omega*, Vol. 35, No. 1, pp. 53–60. DOI: 10.1016/j.omega.2005.03.003

Kalczynski P.J. and Kamburowski J. (2008), An improved NEH heuristic to minimize makespan in permutation flow shops, *Computers and Operations Research*, Vol. 35, No. 9, pp. 3001–3008. DOI: 10.1016/j.cor.2007.01.020

Kalczynski P.J. and Kamburowski J. (2009), An empirical analysis of the optimality rate of flow shop heuristics, *European Journal of Operational Research*, Vol. 198, No. 1, pp. 93–101. DOI: 10.1016/j.ejor.2008.08.021

Liu W. Jin Y. and Price M. (2017), A new improved NEH heuristic for permutation flowshop scheduling problems, *International Journal of Production Economics*, Vol. 193, pp. 21–30. DOI: 10.1016/j.ijpe.2017.06.026

Nagano M.S. and Moccellin, J.V. (2002), A high quality solution constructive heuristic for flow shop sequencing, *Journal of the Operational Research Society*, Vol. 53, No. 12, pp. 1374–1379. DOI: 10.1016/j.scient.2012.10.034

Nawaz M., Enscore Jr E.E. and Ham I. (1983), A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *Omega*, Vol. 11, No. 1, pp. 91–95. DOI: 10.1016/0305-0483(83)90088-9

Rad S. F., Ruiz R. and Boroojerdian N. (2009), New high performing heuristics for minimizing makespan in permutation flowshops, *Omega*, Vol. 37, No. 2, pp. 331–345. DOI: 10.1016/j.omega.2007.02.002

Rossi F.L., Nagano M.S. and Neto R.F.T. (2016), Evaluation of high performance constructive heuristics for the flow shop with makespan minimization, *The International Journal of Advanced Manufacturing Technology*, Vol. 87, pp. 125–136. DOI: 10.1007/s00170-016-8484-9

Ruiz R. and Maroto C. (2005), A comprehensive review and evaluation of permutation flowshop heuristics, *European Journal of Operational Research*, Vol. 165, No. 2, pp. 479–494. DOI: 10.1016/j.ejor.2004.04.017

Semančo P. and Modrák V. (2012), A comparison of constructive heuristics with the objective of minimizing makespan in the flow-shop scheduling problem, *Acta Polytechnica Hungarica*, Vol. 9, No. 5, pp. 177–190.

Taillard E. (1990), Some efficient heuristic methods for the flow shop sequencing problem, European, *Journal of Operational Research*, Vol. 47, No. 1, pp. 65–74. DOI: 10.1016/0377-2217(90)90090-X

Taillard E. (1993), Benchmarks for basic scheduling problems, *European Journal of Operational Research*, Vol. 64, No. 2, pp. 278–285. DOI: 10.1016/0377-2217(93)90182-M

Vallada E., Ruiz R. and Framinan J.M. (2015), New hard benchmark for flowshop scheduling problems minimising makespan, *European Journal of Operational Research*, Vol. 240, No. 3, pp. 666-677. DOI: 10.1016/j.ejor.2014.07.033

Vasiljevic D. and Danilovic M. (2015), Handling ties in heuristics for the permutation flow shop scheduling problem, *Journal of Manufacturing Systems*, Vol. 35, pp. 1–9. DOI: 10.1016/j.jmsy.2014.11.011

Ying K.C. and Lin S.W. (2013), A high-performing constructive heuristic for minimizing makespan in permutation flowshops, *Journal of Industrial and Production Engineering*, Vol. 30, No. 6, pp. 355–362. DOI: 10.1080/21681015.2013.843597