Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 357–361
DOI: 10.15439/2022F103
ISSN 2300-5963 ACSIS, Vol. 30

Improving *N*-NEH+ algorithm by using Starting Point method

Radosław Puka
AGH University of Science and Technology in Kraków,
ul. Gramatyka 10, 30-067 Kraków, Poland
Email: rpuka@zarz.agh.edu.pl

Bartosz Łamasz, Iwona Skalna
AGH University of Science and Technology in Kraków,
ul. Gramatyka 10, 30-067 Kraków, Poland
Email: {blamasz, iskalna}@zarz.agh.edu.pl

***Abstract*—The *N*-NEH+ algorithm is one of the most efficient construction algorithms for solving the permutation flow-shop problem with the makespan criterion. It extends the well-known NEH heuristic with the *N*-list technique. In this paper, we propose the Starting Point (SP) method that employs a new strategy for using the *N*-list technique. The SP method allows to obtain an algorithm that is a combination of NEH and an *N*-list-based algorithm. Extensive numerical experiments on the standard set of Taillard's and VRF benchmarks show that the SP method significantly improves the results (average relative percentage deviation) of the NEH and *N*-NEH+ algorithms.**

I. Introduction

THE permutation flow-shop problem (PFSP) is one of the most famous combinatorial optimization problems in the industry. It can be defined as follows: given two finite sets of m machines $\{M_1, \ldots, M_m\}$ and n jobs $\{J_1, \ldots, J_n\}$, each of which should go through all the m machines in the same order, the goal is to find the ordering of jobs that minimizes the assumed goal function (makespan, total tardiness, flow time, cost, energy consumption, etc.).

The PFSP with makespan criterion, commonly referred to as $Fm|prmu|C_{\max}$ [1], which is our main concern, is undoubtedly the most frequently investigated scheduling problem. Garey and Johnson [2] have shown that $Fm|prmu|C_{\max}$ is NP-hard if $m \geqslant 3$. Therefore, various heuristic algorithms have been developed to solve this problem. One of the most popular algorithms for solving $Fm|prmu|C_{\max}$ is the $\mathcal{O}(n^3m)$ NEH construction heuristic proposed by Navaz, Enscore and Ham [3]. Since 1983 NEH has been commonly regarded as the best heuristic for solving $Fm|prmu|C_{\max}$, and many attempts have been made to improve it. Taillard [4] proposed the so-called *acceleration technique* that reduces the asymptotic time complexity of NEH to $\mathcal{O}(n^2m)$. New criteria were proposed for the initial job classification, e.g., in [5], [6], [7]. The problem of tie-breaking (i.e., how to choose among different subsequences with the same best partial makespan) was considered, e.g., in [8], [9], [10]. One of the most efficient modifications of NEH is the *N*-NEH+ algorithm recently proposed by Puka et al. [11] which, roughly speaking, combines the NEH heuristic with the *N*-list technique [11].

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.

In this paper, we propose the Starting Point (SP) method that is based on a new strategy of using the *N*-list technique. The extensive numerical experiments on the standard Taillard's and VRF benchmarks show that the SP method can significantly improve the results of the *N*-NEH+ algorithm.

II. Starting Point method

The general scheme of the NEH algorithm for solving $Fm|prmu|C_{\max}$ is presented in Algorithm 1.

Algorithm 1 NEH algorithm for solving $Fm|prmu|C_{\max}$

Initial phase: Sort n jobs in non-increasing order of their total processing time and put them into the initial list of jobs $L = \{1, \ldots, n\}$.
Insertion phase: Schedule the first job and remove it from L
for $k = 2, \ldots, n$ **do**
  Insert the job k in the place that minimizes the partial makespan among the k possible ones
  Remove job k from the list.
end for

Ruiz and Maroto [12] compared NEH with 25 heuristics for solving PFSPs and it turned out that NEH performed the best both with respect to the quality of the results and the running time. It is no wonder then that a lot of studies on the improvements of NEH performance have been published in the scientific literature.

One of the most efficient NEH-based algorithms for solving $Fm|prmu|C_{\max}$ is the *N*-NEH+ algorithm [11]. It relies on multiple run of the *N*-NEH algorithm (see Algorithm 2) that extends NEH with the *N*-list technique. More specifically, *N*-NEH+ runs *N*-NEH for the length of the *N*-list ranging in the interval $[0, N]$, where $N \leqslant n - 1$ is the parameter of the *N*-NEH+ algorithm, and takes the best result of all runs. Let us note that if $N = 1$ (i.e., the *N*-list technique is not used) the *N*-NEH+ algorithm is equivalent to NEH.

The Starting Point method proposed in this paper, presented in Algorithm 3, is based on the following strategy: the first N' (N' is a parameter of the SP method) steps are performed as in NEH, and the remaining steps are performed with the use of the *N*-list technique. For example, given a problem with $n = 100$ and $N' = 0.2n$, the first 20 jobs will be scheduled as in NEH and the remaining 80 jobs as in *N*-NEH.

Algorithm 2 N-NEH algorithm for solving $Fm|prmu|C_{\max}$

Initial phase: Sort n jobs in non-increasing order of their total processing time and put them into the list L.
Insertion phase: Initialize the partial sequence $S = \{1\}$.
Initialize the list L_N of N candidate jobs and remove the respective jobs from L.
for $k = 2, \ldots n$ **do**
  Evaluate each job in L_N, put the best job in the respective place in S and remove this job from L_N.
  if $L \neq \emptyset$ **then**
    Append the first job from L to L_N and remove this job from L.
  end if
end for

Algorithm 3 SP method for solving $Fm|prmu|C_{\max}$

Initial phase: Sort n jobs in non-increasing order of their total processing time and put them into the list L.
Insertion phase: Initialize the partial sequence $S = \{1\}$.
for $k = 2, \ldots, \alpha$ **do**
  Insert the job k at the place that minimizes the partial makespan among the k possible ones.
end for
Initialize the list L_N of N candidate jobs and remove the respective jobs from L.
for $k = \alpha + 1, \ldots, n$ **do**
  Evaluate each job in L_N, put the best job in the respective place in S and remove this job from L_N.
  if $L \neq \emptyset$ **then**
    Append the first job from L to L_N and remove this job from L.
  end if
end for

For the purposes of this work, the Starting Point method will be denoted as SP(N')N(N), where N' is a parameter of the SP method and N is the length of the *N*-list. Additionally, SP+(N')N(N) will be used to denote the method that relies on running SP(n')N(N) with n' ranging in the interval $[0, N']$; the result of the SP+(N')N(N) method is the best result out of all runs. Similarly, SP+(N')N+(N) will be used to denote the method that runs the SP(n')N(n'') method with n' ranging in the interval $[0, N']$ and n'' ranging in the interval $[1, N]$. Let us note that the SP(0)N-NEH+ algorithm (0 means that the SP method is not used) is equivalent to the *N*-NEH+ algorithm, whereas SP(N')N+(1) is equivalent to NEH.

Since the optimal solution is not known for some instances, the results obtained by a given algorithm are compared with the best solution known so far. The average relative percentage deviation (ARPD) of an algorithm is given as

$$\mathrm{ARPD} = \frac{1}{I}\sum_{i=1}^{I}\left(C_{\max,i} - Best_i\right)/Best_i, \qquad (1)$$

where I is the number of instances, $C_{\max,i}$ is the solution of the algorithm on the instance i, and $Best_i$ is the best solution known so far for this instance. Additionally, for many computed instances, the computational effort of the evaluated algorithm is measured by using the average CPU time (ACPU) given as:

$$\mathrm{ACPU} = \left(\sum_{i=1}^{I}\mathrm{CPU}_i\right)/I, \qquad (2)$$

where CPU_i is the CPU time of the algorithm on instance i.

The computational experiments that aim to verify the performance of the SP+(N')N+(N) method are presented in the next section. The performance of the method is assessed by using ARPD and ACPU measures.

III. Computational experiment

The performance of the SP(N')N+(N) and SP+(N')N+(N) algorithms (with Taillard's acceleration) was verified on two benchmarks: Taillard's benchmark with 120 instances [13], and VRF benchmark with 240 Small (S) and 240 Large (L) instances [14]. The algorithm was implemented in C# and all the computations were carried out on a computer with two Intel Xeon E5-2660 v4 CPUs (14 cores, each with 2.0 GHz base clock speed).

The results of SP(N')N+(N) (Table I) show that the use of the *N*-list technique after performing N' steps of insertion phase of NEH can improve the results of the *N*-NEH+ algorithm, i.e., the SP(N')N+(N) method can outperform *N*-NEH+. However, if N' is too large ($N' > 0.3n$), the probability of obtaining worse results is greater than in the case of $N' = 0$. The most interesting results are obtained for $N' = 0.1n$ and $N' = 0.2n$. However, it should be noted that improving the results is not possible for all lengths of the *N*-list.

The results of SP+(N')N+(N) (Table II) show that the greatest decrease in the ARPD values can be observed for smaller values of N'. It can also be observed that for $N' = 0.1n$, the average improvement of N-NEH+ results is 5.9% and for $N' = 0.3n$, the average decrease in the ARPD value is greater than 11.6%. For all the considered benchmarks, a regularity regarding the effectiveness of using the SP+(N')N+(N) method can also be noticed: the longer the *N*-list, the higher the average percentage decrease in the ARPD value.

From Table III it can be concluded that the larger N', the shorter computational time of the SP(N')N+(N) algorithm. The computational time also decreases when increasing the length of the *N*-list. The average decrease in computational time between SP(0)N+(2) (i.e., N+(2)) and SP(0.9n)N+(2) ranges from 14% to 16%, and the average decrease in computational time between SP(0)N+(16) (i.e., N+(16)) and SP(0.9n)N+(16) ranges from 63% to 69%. Table IV shows that the computational time of SP+(N')N+(N) increases both with N and N', however the increase with N is much faster.

The ARPD and ACPU values are also shown in Figures 1–3, where the y-axis represents ARPD and x-axis represents ACPU (in logarithmic scale) of the SP+(N')N+(N), NEH

TABLE I
APRD[%] FOR SP(N')N+(N) METHOD ON TAILLARD, VRF S AND VRF L INSTANCES; BEST VALUES FOR EACH $N > 1$ ARE MARKED IN BOLD

Bench.	N	N'									
		0	$0.1n$	$0.2n$	$0.3n$	$0.4n$	$0.5n$	$0.6n$	$0.7n$	$0.8n$	$0.9n$
Tai	1	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33
	2	2.95	2.96	**2.93**	2.97	2.96	2.93	2.96	3.07	3.10	3.27
	4	2.55	2.65	**2.55**	2.59	2.59	2.62	2.69	2.87	2.99	3.22
	8	2.32	2.33	**2.29**	2.35	2.37	2.41	2.52	2.68	2.91	3.18
	16	2.20	**2.16**	2.17	2.21	2.24	2.29	2.39	2.60	2.86	3.16
VRF S	1	3.84	3.84	3.84	3.84	3.84	3.84	3.84	3.84	3.84	3.84
	2	3.32	3.32	**3.31**	3.40	3.47	3.45	3.54	3.63	3.71	3.80
	4	2.95	**2.94**	2.99	3.03	3.09	3.19	3.34	3.41	3.59	3.78
	8	**2.64**	2.66	2.67	2.73	2.85	3.01	3.14	3.31	3.53	3.78
	16	2.47	**2.42**	2.48	2.56	2.70	2.86	3.06	3.27	3.53	3.78
VRF L	1	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33	3.33
	2	3.00	**3.00**	3.01	3.01	3.03	3.04	3.08	3.11	3.15	3.21
	4	2.67	**2.65**	2.68	2.72	2.72	2.76	2.84	2.89	2.99	3.11
	8	2.39	**2.36**	2.40	2.41	2.44	2.52	2.59	2.69	2.83	3.02
	16	2.14	**2.11**	2.12	2.14	2.20	2.28	2.36	2.50	2.69	2.95

TABLE II
APRD[%] FOR SP+(N')N+(N) METHOD ON TAILLARD, VRF S AND VRF L INSTANCES

Bench.	N	N'								
		$0.1n$	$0.2n$	$0.3n$	$0.4n$	$0.5n$	$0.6n$	$0.7n$	$0.8n$	$0.9n$
Tai	2	2.82	2.74	2.66	2.63	2.58	2.50	2.49	2.48	2.47
	4	2.41	2.31	2.23	2.19	2.15	2.12	2.12	2.11	2.11
	8	2.15	2.04	1.98	1.94	1.93	1.91	1.89	1.89	1.89
	16	2.02	1.90	1.85	1.82	1.81	1.79	1.78	1.78	1.78
VRF S	2	3.15	3.01	2.93	2.88	2.84	2.81	2.80	2.78	2.78
	4	2.74	2.62	2.54	2.50	2.47	2.46	2.44	2.43	2.43
	8	2.45	2.32	2.24	2.20	2.19	2.18	2.17	2.16	2.16
	16	2.24	2.12	2.04	2.02	2.00	2.00	1.99	1.99	1.99
VRF L	2	2.89	2.84	2.80	2.77	2.74	2.73	2.73	2.72	2.72
	4	2.55	2.51	2.49	2.46	2.45	2.45	2.45	2.45	2.45
	8	2.28	2.24	2.22	2.21	2.20	2.20	2.19	2.19	2.19
	16	2.05	2.00	1.98	1.97	1.97	1.97	1.96	1.96	1.96

TABLE III
ACPU[MS] OF SP(N')N+(N) METHOD ON TAILLARD, VRF S AND VRF L INSTANCES

Bench.	N	N'									
		0	$0.1n$	$0.2n$	$0.3n$	$0.4n$	$0.5n$	$0.6n$	$0.7n$	$0.8n$	$0.9n$
Tai	1	38.5	36.3	36.3	36.3	36.2	36.9	36.2	36.2	36.4	36.8
	2	89.9	87.1	86.6	85.7	85.0	84.2	81.8	79.8	78.1	75.7
	4	235.8	231.9	229.2	225.2	218.2	209.5	200.4	187.5	175.1	160.6
	8	702.6	691.8	678.5	656.7	624.9	585.7	541.7	486.8	430.6	358.1
	16	2304.1	2267.4	2205.8	2107.5	1979.7	1810.0	1614.7	1384.1	1129.0	832.8
VRF S	1	1.4	1.3	1.3	1.2	1.2	1.2	1.2	1.2	1.2	1.2
	2	3.0	2.9	2.9	2.8	2.8	2.7	2.7	2.6	2.5	2.4
	4	7.6	7.5	7.3	7.1	6.8	6.6	6.3	5.8	5.4	4.9
	8	21.5	21.0	20.4	19.7	18.7	17.3	15.8	13.8	11.9	9.9
	16	63.4	62.2	59.8	56.4	52.0	46.4	39.9	32.3	25.2	19.7
VRF L	1	625.5	619.0	618.5	617.5	616.4	616.6	616.2	619.2	615.7	616.7
	2	1488.2	1477.7	1468.2	1452.0	1435.0	1413.1	1385.4	1354.5	1314.8	1276.2
	4	3959.7	3922.1	3868.1	3782.5	3678.1	3539.8	3377.0	3186.0	2965.1	2722.4
	8	11838.0	11714.1	11473.0	11072.6	10566.2	9927.0	9149.9	8261.5	7242.4	6107.0
	16	39163.6	38688.9	37624.2	35965.8	33753.0	30997.0	27669.3	23850.2	19489.1	14607.9

TABLE IV
ACPU[MS] OF SP+(N')N+(N) METHOD ON TAILLARD, VRF S AND VRF L INSTANCES

Bench.	N	N'								
		$0.1n$	$0.2n$	$0.3n$	$0.4n$	$0.5n$	$0.6n$	$0.7n$	$0.8n$	$0.9n$
Tai	2	177.1	263.6	349.3	434.3	518.5	600.2	680.0	758.1	833.8
	4	467.8	697.0	922.2	1140.4	1349.9	1550.3	1737.8	1912.9	2073.5
	8	1394.4	2072.9	2729.7	3354.6	3940.3	4482.0	4968.8	5399.4	5757.5
	16	4571.5	6777.3	8884.8	10864.4	12674.4	14289.1	15673.2	16802.1	17634.9
VRF S	2	6.0	8.9	11.6	14.4	17.1	19.8	22.4	24.9	27.3
	4	15.1	22.4	29.5	36.4	43.0	49.2	55.0	60.4	65.3
	8	42.5	62.9	82.6	101.3	118.7	134.5	148.3	160.1	170.1
	16	125.6	185.3	241.7	293.7	340.1	380.0	412.3	437.5	457.2
VRF L	2	2965.9	4434.0	5886.1	7321.1	8734.2	10119.6	11474.1	12788.8	14065.1
	4	7881.8	11749.8	15532.3	19210.4	22750.2	26127.1	29313.2	32278.2	35000.6
	8	23552.1	35025.1	46097.7	56663.9	66590.9	75740.8	84002.3	91244.7	97351.7
	16	77852.5	115476.7	151442.5	185195.5	216192.5	243861.8	267712.0	287201.0	301809.0

and N-NEH+ algorithms. As can be seen from the figures, for all benchmarks, it is difficult to indicate the parameters for which the SP+ method is dominated by other algorithm. In most cases, the usage of SP+(N')N+(2) does not allow obtaining non-dominated results. The results for the remaining parameters form the Pareto front.

Table V presents the comparison of the APRD and ACPU values of the selected algorithms for solving $Fm|prmu|C_{\max}$(cf. [15]). It is not hard to see that these results confirm the good performance of the SP+(N')N+(N) algorithm. This means that the new strategy of delaying the usage of the N-list allows to improve the results of N-NEH+ and makes the SP+(N')N+(N) even more competitive with FRB algorithms [16]. One may argue that in most cases the SP+(N')N+(N) algorithm is more time consuming than FRB algorithms; however, you should keep in mind that SP+(N')N+(N) is (unlike FRB) a construction algorithm, and what is more, it allows you to obtain the entire population of solutions, not just one solution as in the case of FRB.

TABLE V
ARPD[%] AND ACPU[MS] VALUES OF SELECTED ALGORITHMS ON TAILLARD'S BENCHMARK.

Algorytm	ARPD	ACPU	Algorytm	ARPD	ACPU
RAER	3.89	132.9	$FRB4_2$	2.33	235.2
RAER-di	3.53	277.5	N+(8)	2.32	702.6
NEH	3.33	38.5	SP+(0.2)N+(4)	2.31	697.0
NEMR	3.16	215.1	SP+(0.3)N+(4)	2.23	922.2
KKER	3.15	127.1	N+(16)	2.20	2304.1
NEHKK1-di	3.15	77.6	SP+(0.4)N+(4)	2.19	1140.4
NEH1-di	3.11	77.9	SP+(0.1)N+(8)	2.15	1394.4
NEHKK2	3.09	39.5	$FRB4_4$	2.13	379.5
NEHR	3.05	133.4	SP+(0.2)N+(8)	2.04	2072.9
NEH-di	3.03	76.9	SP+(0.1)N+(16)	2.02	4571.5
CL_WTS	3.02	1789.9	SP+(0.3)N+(8)	1.98	2729.7
NEMR-di	2.97	386.6	$FRB4_8$	1.95	639.3
N+(2)	2.95	89.9	SP+(0.4)N+(8)	1.94	3354.6
NEHFF	2.9	42.2	FRB2	1.93	1335.3
KKER-di	2.86	261.2	$FRB4_6$	1.91	511.2
NEHR-di	2.85	274.5	SP+(0.2)N+(16)	1.90	6777.3
NEHD-di	2.84	344.4	$FRB4_{10}$	1.87	765.0
SP+(0.1)N+(2)	2.82	177.1	SP+(0.3)N+(16)	1.85	8884.8
SP+(0.2)N+(2)	2.74	263.6	SP+(0.4)N+(16)	1.82	10864.4
SP+(0.3)N+(2)	2.66	349.3	$FRB4_{12}$	1.79	879.5
SP+(0.4)N+(2)	2.63	434.3	FRB3	1.61	10522.9
N+(4)	2.55	235.8	FRB5	1.48	30207.4
SP+(0.1)N+(4)	2.41	467.8			

IV. CONCLUSION

This work proposes a Starting Point method that improves N-list technique-based algorithms for solving the permutation flow-shop problem with makespan criterion. The general idea behind the proposed method is to delay the usage of the N-list technique. The extensive numerical experiments on the standard Taillard's and VRF benchmarks show that for both benchmarks the Starting Point method significantly improves the results of the N-NEH and N-NEH+ algorithms. According to the best knowledge of the authors, the analyzed SP+(N')N+(N) algorithm allows obtaining the best results among the results of existing construction algorithms. It is also worth noting that, unlike the more efficient FRB algorithm (which is not construction algorithm), the SP+(N')N+(N) algorithm can return not just one but multiple complete rankings. This issue is important because the results of the SP+(N')N+(N) algorithm may be used as the initial population for genetic algorithms. The last important aspect of the SP+(N')N+(N) algorithm is the ease of parallelization of the computations (e.g., for different values of N' and N). For example, the computation time of SP+(0.3n)N(N) run is parallel on 4 cores (one core = one N' value), is equal to the computational time of N-NEH for the same length of the N-list. At the same time, the improvement in the ARPD value for different benchmarks ranges from 6.8% to even 17.2%.

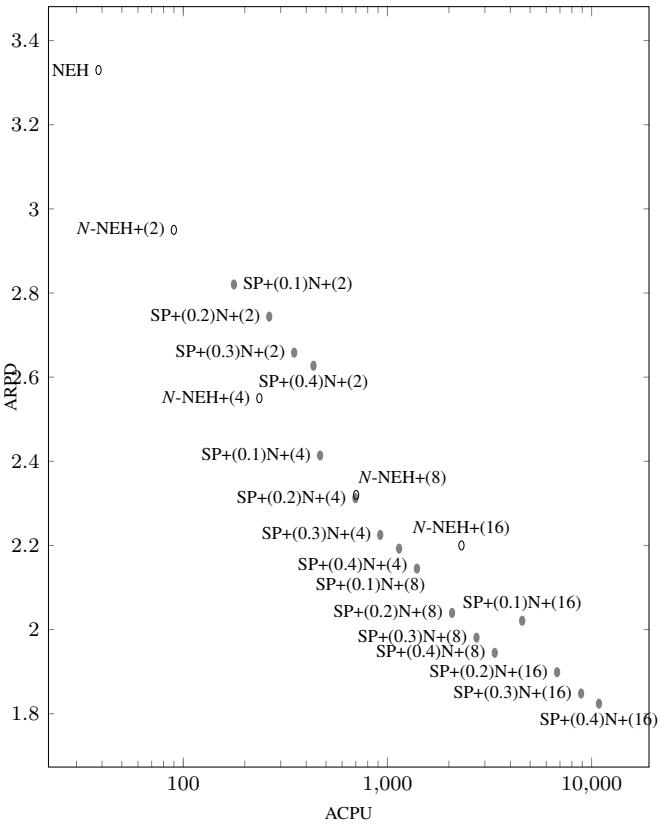

Fig. 1. ARPD vs. ACPU (in logarithmic scale) of SP+(N')N+(N) algorithm on Taillard's instances.

REFERENCES

[1] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 287–326. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016750600870356X

[2] M. Garey and D. Johnson, *Computers and intractability*. San Francisco: W.H. Freeman, 1979, vol. 174.

[3] M. Nawaz, E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0305048383900889

[4] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 47, no. 1, pp. 65–74, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037722179090090X

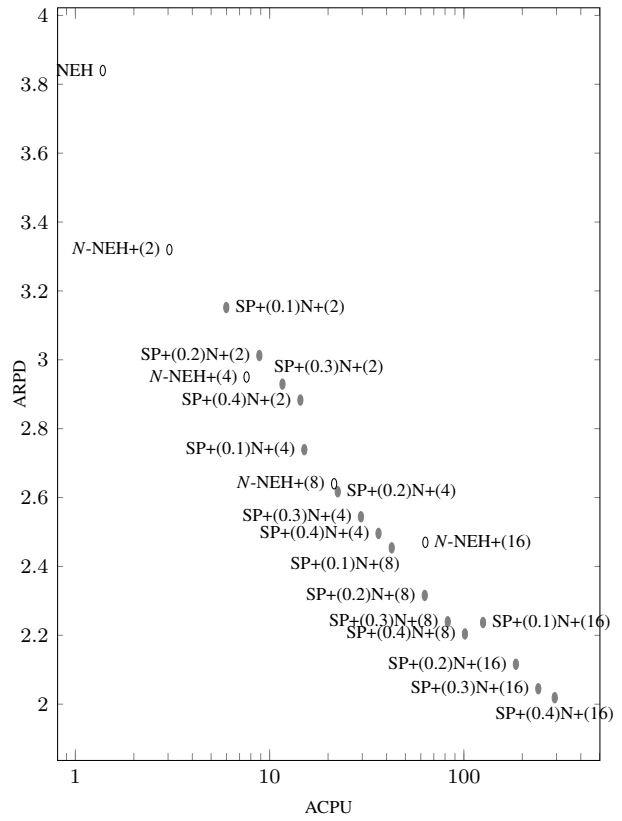

Fig. 2. ARPD vs. ACPU (in logarithmic scale) of SP+(N')N+(N) algorithm on VRF Small instances.

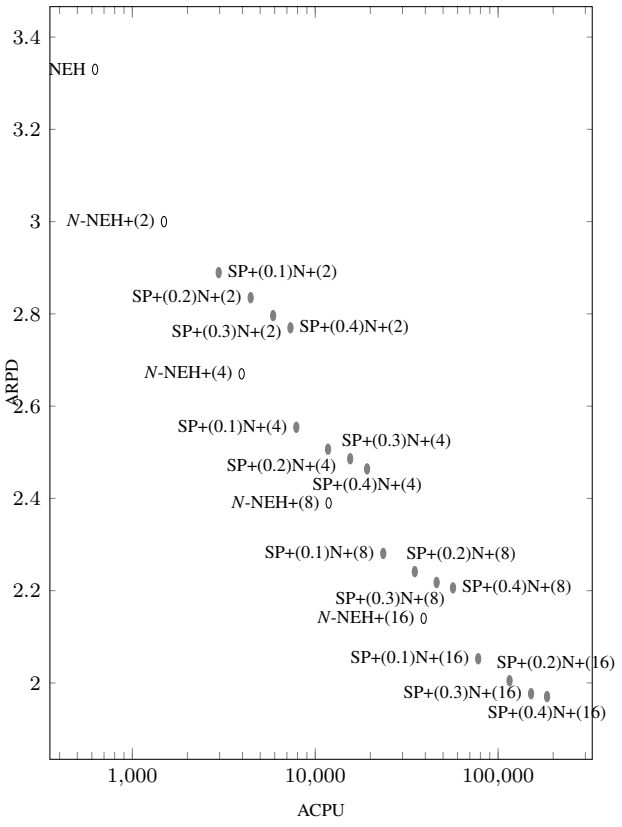

Fig. 3. ARPD vs. ACPU (in logarithmic scale) of SP+(N')N+(N) algorithm on VRF Large instances.

[5] P. Kalczynski and J. Kamburowski, "On the NEH heuristic for minimizing the makespan in permutation flow shops," *Omega*, vol. 35, no. 1, pp. 53–60, 2007.

[6] ——, "An improved NEH heuristic to minimize makespan in permutation flow shops," *Computers & Operations Research*, vol. 35, no. 9, pp. 3001–3008, 2008, part Special Issue: Bio-inspired Methods in Combinatorial Optimization.

[7] M. Nagano and J. Moccellin, "A high quality solution constructive heuristic for flow shop sequencing," *Journal of the Operational Research Society*, vol. 53, no. 12, pp. 1374–1379, 2002.

[8] P. Kalczynski and J. Kamburowski, "An empirical analysis of the optimality rate of flow shop heuristics," *European Journal of Operational Research*, vol. 198, no. 1, pp. 93–101, 2009.

[9] V. Fernandez-Viagas and J. Framinan, "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem," *Computers & Operations Research*, vol. 45, pp. 60–67, 2014.

[10] I. Ribas, R. Companys, and X. Tort-Martorell, "Comparing three-step heuristics for the permutation flow shop problem," *Computers & Operations Research*, vol. 37, no. 12, pp. 2062–2070, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030505481000050X

[11] R. Puka, J. Duda, A. Stawowy, and I. Skalna, "N-neh+ algorithm for solving permutation flow shop problems," *Computers & Operations Research*, vol. 132, p. 105296, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054821000885

[12] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494, 2005.

[13] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993, project Management anf Scheduling. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037722179390182M

[14] E. Vallada, R. Ruiz, and J. Framinan, "New hard benchmark for flowshop scheduling problems minimising makespan," *European Journal of Operational Research*, vol. 240, no. 3, pp. 666–677, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221714005992

[15] V. Fernandez-Viagas, R. Ruiz, and J. Framinan, "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, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221716308074

[16] S. Rad, R. Ruiz, and N. Boroojerdian, "New high performing heuristics for minimizing makespan in permutation flowshops," *Omega*, vol. 37, no. 2, pp. 331–345, 2009.