DOI: 10.20535/kpisn.2020.4.209129
UDC 519.161+519.687.1+004.023

V.V. Romanuke*
Polish Naval Academy, Gdynia, Poland
*corresponding author: romanukevadimv@gmail.com

TIGHT-TARDY PROGRESSIVE IDLING-FREE 1-MACHINE PREEMPTIVE SCHEDULING WITH JOB PRIORITY WEIGHTS BY HEURISTIC'S EFFICIENT JOB ORDER INPUT

Background. In setting a problem of minimizing total weighted tardiness by the heuristic based on remaining available and processing periods, there are two opposite ways to input the data: the job release dates are given in either ascending or descending order. It was recently ascertained that scheduling a few equal-length jobs is expectedly faster by ascending order, whereas scheduling 30 to 70 equal-length jobs is 1.5 % to 2.5 % faster by descending order. For the number of equal-length jobs between roughly 90 and 250, the ascending job order again results in shorter computation times. In the case when the jobs have different lengths, the significance of the job order input is much lower. On average, the descending job order input gives a tiny advantage in computation time. This advantage decreases as the number of jobs increases.

Objective. The goal is to ascertain whether the job order input is significant in scheduling by using the heuristic for the case when the jobs have different lengths with job priority weights. Job order efficiency will be studied on tight-tardy progressive idling-free 1-machine preemptive scheduling.

Methods. To achieve the said goal, a computational study is carried out with a purpose to estimate the averaged computation time for both ascending and descending orders of job release dates. First, the computation time for the ascending job order input is estimated for a series of job scheduling problems. Then, in each instance of this series, job lengths, priority weights, release dates, and due dates are reversed making thus the respective instance for the descending job order input, for which computation time is estimated as well.

Results. The significance of the job order input is much lower than that for the case of jobs without priorities. With assigning the job priority weights, the job order input becomes further "dithered", adding randomly scattered priority weights to randomly scattered job lengths and partially randomized due dates. On average, the descending job order input is believed to give a tiny advantage in computation time in scheduling up to 100 jobs. However, this advantage, if any (being tinier than that in the case of random job lengths without priorities), quickly vanishes as the number of jobs increases.

Conclusions. It is better to compose job scheduling problems which would be closer to the case with equal-length jobs without priorities, where the saved computational time can be counted in hours. Even if the job lengths and priority weights are scattered, it is recommended to artificially "flatten" them. When artificial manipulations over job processing periods and job priority weights are impossible, it is recommended to use the descending job order input in scheduling up to 100 jobs, and either job order input in scheduling more than 100 jobs, although substantial benefits are not expected in this case.

Keywords: preemptive single machine job scheduling; total weighted tardiness; heuristic; ascending job order; descending job order; computation time; efficient job order.

Introduction

Job scheduling is an important combinatorial problem whose practical impact is pretty intense. Tardiness is one of the main features in scheduling. The exact minimization of total weighted tardiness is possible just for a few jobs whose processing periods are not very long [1]. Heuristics are the only means which capable of scheduling hundreds and thousands of jobs, or more [2, 3]. In some practical tasks, moreover, the entire schedule can be an extremely long sequence of jobs [4], whereas the heuristics allow online scheduling (once a job is scheduled at a time moment, it will not be changed and thus the jobs already scheduled can be executed straightforwardly without waiting for the entire schedule) [5].

The heuristic based on the remaining available period and remaining processing period [6] is closely the best one. Its efficient job order input (in ascending or descending order) was studied in articles [7] and [5]. The efficiency implied faster computations.

Article [7] ascertained that, in scheduling by using the heuristic, the job order input is significant for the case of tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs. Scheduling a few jobs is expectedly faster by ascending order, although there were many computational artifacts [5]. Article [7] showed that scheduling 30 to 70 jobs is 1.5 % to 2.5 % faster by descending order. However, scheduling up to 90 jobs is expectedly still faster by descending order, although a risk of losing this advantage exists. For the number of jobs between roughly 90 and 250, the ascending job order again results in shorter computation times. Since the point of about 250 jobs, the advantage trend (of either ascending or descending order) appears more stable. Besides, the average relative difference does not exceed 1.5 % for 2 to 1000 jobs consisting up to 17 processing periods. Article [7] also revealed that, for obtaining a statistically reliable computation speed advantage, it is better to consider no less than 250 jobs. However, as either the number of jobs or the number of job parts increases, the computation speed advantage may become unstable and eventually vanish. In the same time, in the case of scheduling at least a few thousand jobs having just a few processing periods each, the ascending job order can save a lot of computational time — after solving thousands of such cases the saved time may be counted in hours (see computational examples in [7]).

In the case when the jobs have different lengths (i. e., whose number of processing periods varies) studied in [5], the significance of the job order input is much lower than that for the case of equal-length jobs. On average, the descending job order input gives a tiny advantage in computation time. This advantage decreases as the number of jobs increases. The decrement resembles a steep exponential decrease. The factual advantage is so insignificant that even after solving long series of job scheduling problems the saved computational time cannot be counted in minutes, not speaking about hours. Theoretically, the heuristic's efficient job order input does exist but its efficiency can be practically used only by working on extremely long series of scheduling problems where the number of jobs should not exceed 300 [5].

The research is to be finalized with the case when the jobs have different priority weights. This is an extension (continuation) of the cases studied in [7] and [5], where the priorities are not sorted. Now, the influence of the generalized tight-tardy progressive idling-free 1-machine preemptive scheduling on the heuristic's computational times (for both the ascending and descending job order) is to be studied.

Problem statement

The goal is to ascertain whether the job order input is significant in scheduling by using the heuristic for the case of tight-tardy progressive idling-free 1-machine preemptive scheduling with job priority weights. The five following tasks will be fulfilled for achieving this goal. First of all, the principles of the ascending and descending job order inputs are stated, whereupon the heuristic is shortly stated for the case of when the jobs have different lengths and different priority weights. Then, generation of the job scheduling problem instances is stated for this case, whereupon a computational study is carried out for estimating the relative difference between averaged computation times for both the ascending and descending job order. Finally, a conclusion is made on whether the efficient job order input exists for minimizing total weighted tardiness by the heuristic (based on remaining available and processing periods). Besides, the final comparison of the respective results for the three classes of job scheduling problems (total tardiness by equal job lengths [7], total tardiness by different job lengths [5], total weighted tardiness) should be made. These results are to be arranged and the corresponding recommendations are to be formulated.

Principles of the ascending and descending job order inputs

In minimizing total weighted tardiness, the initial data are job lengths, job release dates, priority weights, and due dates [8]. There are two opposite ways to input the data. On one hand, the job release dates are given in ascending order. For N jobs, $N \in \mathbb{N} \setminus \{1\}$, without losing generality, the ascending job order input (this is the common way of inputting the data) corresponds to due dates

$$d_n = r_n + H_n - 1 + b_n \quad \forall n = \overline{1, N} \tag{1}$$

by the respective release date r_n of job n, its length H_n and a random due date (1) shift

$$b_n = \psi(H_n \cdot \zeta) \quad \text{for} \quad n = \overline{1, N} \tag{2}$$

with a pseudorandom number ζ drawn from the standard normal distribution (with zero mean and unit variance), and function $\psi(\xi)$ returning the integer part of number ξ (e. g., see [1, 5, 7]). Job n has priority weight w_n (which, like the others, is a positive integer). In particular, the release dates can be given in ascending order as follows [5, 7]:

$$r_n = n \quad \forall n = \overline{1, N}\,. \tag{3}$$

Strictly speaking, the release dates can be permuted as one likes or needs to satisfy some external conditions (obviously, the job processing periods, job priority weights, and due dates are permuted with respect to the release date permutation). Thus, on the other hand, the job release dates are given in descending order as

$$r_n = N - n + 1 \quad \forall n = \overline{1, N} \tag{4}$$

and the descending job order input corresponds to due dates

$$d_n = r_n + H - 1 + b_{N-n+1} \quad \forall n = \overline{1, N}. \tag{5}$$

Due date shifts (2) are generated until

$$d_n \geq 1 \quad \forall n = \overline{1, N}. \tag{6}$$

If simultaneously

$$\begin{gathered} H_n \leq H_{n+1} \text{ and } d_n \leq d_{n+1} \\ \text{and } w_n \geq w_{n+1} \ \forall n = \overline{1, N-1} \end{gathered} \tag{7}$$

for the ascending job order input with (3) and (1), then due date shifts (2) are re-generated as well. So, if one of the inequalities in (7) is violated, then the due dates are given properly for the ascending job order input:

$$d_n = n + H_n - 1 + b_n \quad \forall n = \overline{1, N}. \tag{8}$$

Symmetrically, if simultaneously

$$\begin{gathered} H_n \geq H_{n+1} \text{ and } d_n \geq d_{n+1} \\ \text{and } w_n \leq w_{n+1} \ \forall n = \overline{1, N-1} \end{gathered} \tag{9}$$

for the descending job order input with (4) and (5), then due date shifts (2) are re-generated also. If one of the inequalities in (9) is violated, then the due dates are given properly for the descending job order input:

$$d_n = N - n + H_n + b_{N-n+1} \quad \forall n = \overline{1, N}. \tag{10}$$

Thus, due dates (8) are not given in non-descending order if the job lengths and their priorities have been occasionally generated in non-descending order and in non-ascending order, respectively. This is done so because in the case of when all inequalities (7) are simultaneously true, a schedule ensuring the exactly minimal total weighted tardiness is found trivially, without resorting to any algorithm or model (see Theorem 1 in [9] and [10]). By symmetrical reasoning, due dates (10) are not given in non-ascending order if both the job lengths and their priorities have been occasionally generated in non-ascending order and in non-descending order, respectively: if all inequalities (9) are simultaneously true, an optimal schedule is found trivially as well owing to Theorem 2 in [9] and [10].

The heuristic based on remaining available and processing periods

The basis of the heuristic operating on remaining available and processing periods was firstly introduced in [6] and then developed for minimization of total weighted tardiness in [5, 7, 11, 12]. The heuristic builds stepwise a schedule $\tilde{\mathbf{S}} = [\tilde{s}_t]_{1 \times T}$ as time t progresses up to $T = \sum_{n=1}^{N} H_n$. Before the start,

$$q_n = H_n \quad \forall n = \overline{1, N}. \tag{11}$$

Then, for every set of available jobs

$$A(t) = \{i \in \{\overline{1, N}\} : r_i \leq t \text{ and } q_i > 0\} \subset \{\overline{1, N}\} \tag{12}$$

the remaining available period is

$$b_i = \max\{0, d_i - t + 1\} \quad \forall i \in A(t) \tag{13}$$

and a subset

$$A^*(t) = \arg\max_{i \in A(t)} \frac{w_i}{\max\{q_i, b_i\}} \tag{14}$$

is determined. If $|A^*(t)| = 1$, where

$$A^*(t) = \{i^*\} \subset A(t) \subset \{\overline{1, N}\},$$

then

$$\tilde{s}_t = i^* \text{ by } q_{i^*}^{(\text{obs})} = q_{i^*} \text{ and } q_{i^*} = q_{i^*}^{(\text{obs})} - 1; \tag{15}$$

otherwise

$$A^*(t) = \{i_l^*\}_{l=1}^{L} \subset A(t) \subset \{\overline{1, N}\} \text{ by } L > 1, \tag{16}$$

whence

$$\tilde{s}_t = i_1^* \text{ by } q_{i_1^*}^{(\text{obs})} = q_{i_1^*} \text{ and } q_{i_1^*} = q_{i_1^*}^{(\text{obs})} - 1. \tag{17}$$

Assignment (17) executed by condition (16) for subset (14) implies that, in a case when there are two or more maximal decisive ratios in (14), the earliest job is preferred to be scheduled [7]. Thus, job n is completed after moment $\tilde{\theta}(n; H_n)$ if

$$\tilde{s}_{\tilde{\theta}(n;\, h_n)} = n \quad \forall h_n = \overline{1, H_n} \text{ by } \tilde{\theta}(n; h_n) \in \{\overline{1, T}\}$$

$$\text{and } \tilde{\theta}(n; h_n) < \tilde{\theta}(n; h_n + 1) \text{ for } h_n = \overline{1, H_n - 1}$$

in schedule $\tilde{\mathbf{S}} = [\tilde{s}_t]_{1\times T}$ returned by the heuristic. Finally, amount

$$\tilde{\vartheta}(N) = \sum_{n=1}^{N} w_n \cdot \max\{0,\ \tilde{\theta}(n;\ H_n) - d_n\} \qquad (18)$$

is an approximately minimal total weighted tardiness that corresponds to this schedule.

Generation of the job scheduling problem instances

The jobs having different lengths are randomly generated by a method suggested in [5, 11]:

$$H_n = \psi(16\upsilon + 2) \quad \text{for} \quad n = \overline{1,\ N} \qquad (19)$$

with a pseudorandom number υ drawn from the standard uniform distribution on the open interval $(0;\ 1)$. So, the job length is randomly generated between 2 and 17 [5, 7]. The priority weight of job n is randomly generated in the identical way:

$$w_n = \psi(100\upsilon + 1) \quad \text{for} \quad n = \overline{1,\ N}\,. \qquad (20)$$

When job lengths (19), priority weights (20), and due date shifts (2) are properly generated by some N for the ascending job order input, i. e. inequality (6) holds and at least one of the inequalities in (7) is violated, then an ascending order schedule by job lengths $\{H_n\}_{n=1}^{N}$, their priority weights $\{w_n\}_{n=1}^{N}$, release dates (3), and due dates (8) is computed by the heuristic with statements (11) — (18). Alternatively, a descending order schedule by job lengths

$$\begin{aligned} &\{H_n\}_{n=1}^{N} \ \text{ after } \ H_j^{(\text{obs})} = H_j \quad \forall\, j = \overline{1,\ N} \\ &\text{and } H_n = H_{N-n+1}^{(\text{obs})} \ \text{ for } \ n = \overline{1,\ N}, \end{aligned} \qquad (21)$$

priority weights

$$\begin{aligned} &\{w_n\}_{n=1}^{N} \ \text{ after } \ w_j^{(\text{obs})} = w_j \quad \forall\, j = \overline{1,\ N} \\ &\text{and } w_n = w_{N-n+1}^{(\text{obs})} \ \text{ for } \ n = \overline{1,\ N}, \end{aligned} \qquad (22)$$

release dates (4), and due dates (10) is computed as well. In fact, job lengths (21), priority weights (22), release dates (4), and due dates (10) for the descending job order input are obtained by just reversing (i. e., flipping the left and right) job lengths $\{H_n\}_{n=1}^{N}$, priority weights $\{w_n\}_{n=1}^{N}$, release dates (3), and due dates (8) for the ascending job order input [5].

At a fixed number of jobs N and for a job scheduling problem instance tagged by an integer c, denote the schedule computation times by ascending order and descending order by $\tau_{Asc}(N,\ c)$ and $\tau_{Desc}(N,\ c)$ in milliseconds (ms), respectively. If the total number of the instances is C, then the respective averaged computation times for scheduling N jobs are [5]

$$\tau_{Asc}(N) = \frac{1}{C}\sum_{c=1}^{C} \tau_{Asc}(N,\ c) \qquad (23)$$

and

$$\tau_{Desc}(N) = \frac{1}{C}\sum_{c=1}^{C} \tau_{Desc}(N,\ c)\,. \qquad (24)$$

In percentage terms, the relative difference between computation times (23) and (24) is

$$\eta(N) = 100 \cdot \frac{\tau_{Asc}(N) - \tau_{Desc}(N)}{\tau_{Asc}(N)} \ \text{ by } \ N = \overline{2,\ 1000}. \qquad (25)$$

Relative difference (25) will be estimated for $C = 350$ and $C = 450$. Generating more instances will not make the estimation more effective, although even 350 instances at a fixed number of jobs here is superfluous.

Computational study

Just as in articles [7] and [5], the computational study is executed on CPU Intel Core i5-7200U@2.50 GHz using MATLAB R2018a. Relative difference (25) is shown in Fig. 1 for $C = 350$ whose average relative difference

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.3958 \qquad (26)$$

indicates a tiny advantage of the descending job order input. It is still uncertain whether such a tiny advantage could be claimed statistically reliable, although running average relative difference

$$\tilde{\eta}(N) = \frac{1}{N-1} \cdot \sum_{m=2}^{N} \eta(m) \quad \text{for} \quad N = \overline{2,\ 1000} \qquad (27)$$

whose plot is added to the relative difference polyline appears to stand for the descending job order input (owing to the horizontal zero level line put on the plot).

Fig. 2 contains the results of the same computational experiment repeated by another randomizer [13, 14]. Now, average relative difference

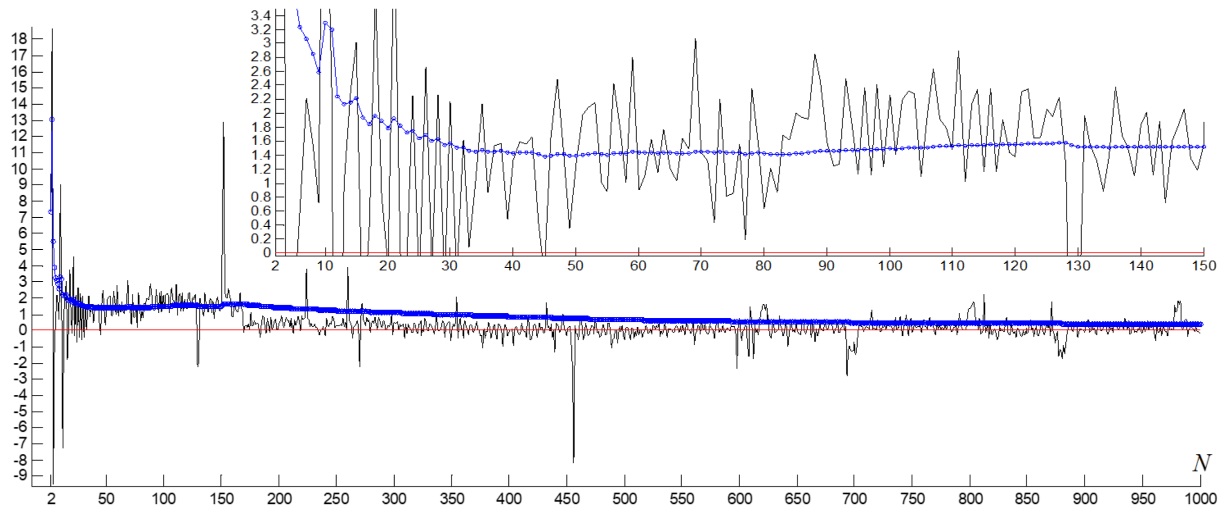

Fig. 1. Relative difference (25) and running average relative difference (27) for $C = 350$: ——— $-\ \eta(N)$; —○— $-\ \tilde{\eta}(N)$

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.2878 \qquad (28)$$

is even lesser than average relative difference (26) by the first version of the computational experiment. However, running average relative difference (27) here appears to be a little bit more persuasive arguing in favor of the descending job order input.

The results of the third computational experiment executed for $C = 450$ are shown in Fig. 3. Here, average relative difference

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.2383 \qquad (29)$$

is slightly lesser than average relative difference (28) by the second version of the computational experiment. Running average relative difference (27) here is similar to those in Fig. 1 and Fig. 2 but it still cannot confirm the statistical reliability of that the descending job order input is faster.

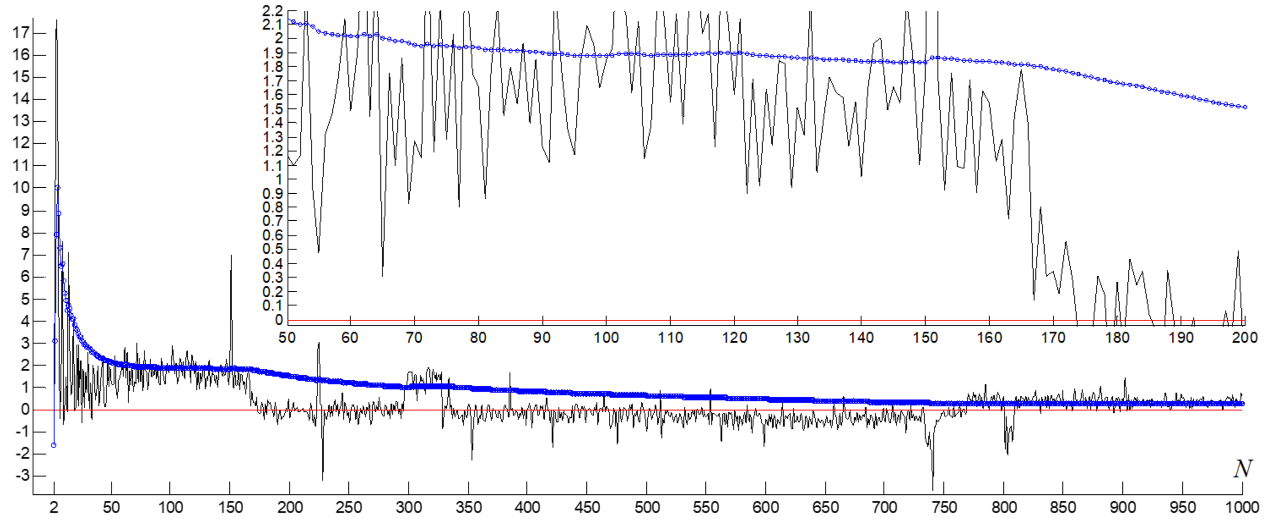

Fig. 2. The second version of relative difference (25) and running average relative difference (27) for $C = 350$: ——— $-\ \eta(N)$; —○— $-\ \tilde{\eta}(N)$

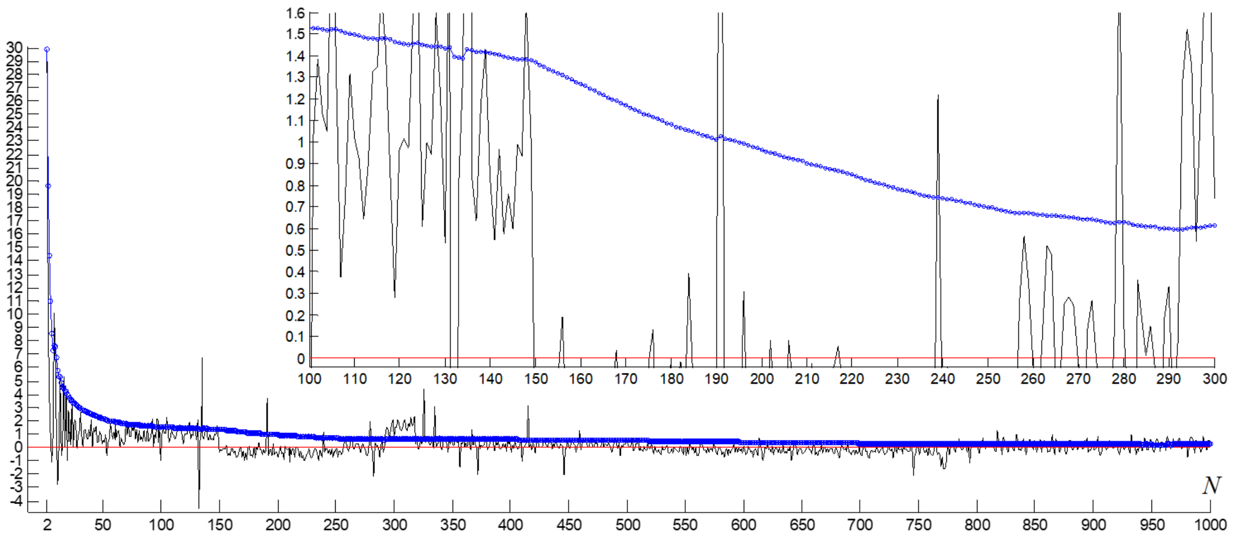

Fig. 3. Relative difference (25) and running average relative difference (27) for $C = 450$: —— – $\eta(N)$; —⊖— – $\tilde{\eta}(N)$

Another way to process the obtained data is to count the sliding average [15, 16] of relative difference (25):

$$\tilde{\eta}([N_k;\, N_{k+1}]) = \frac{1}{N_{k+1} - N_k + 1} \cdot \sum_{m=N_k}^{N_{k+1}} \eta(m) \quad (30)$$

$$\text{for } k = \overline{1, K} \text{ by } N_1 = 2 \text{ and } N_{K+1} = 1000,$$

where

$$N_k < N_{k+1} \text{ by } k = \overline{1, K}$$
$$\text{and } \bigcup_{k=1}^{K} [N_k;\, N_{k+1}] = [N_1,\, N_{K+1}]. \quad (31)$$

Sliding average (30) has a sliding window as an interval $[N_k;\, N_{k+1}]$ which, in general, can be slightly varied on the way to the right endpoint N_{K+1}. Consider three versions of the sliding window: with the length of 20 which is

$$\tilde{\eta}([2;\, 20]),\ \tilde{\eta}([20 \cdot (k-1) + 1;\, 20 \cdot k]) \text{ for } k = \overline{2, 50} \quad (32)$$

(the very first interval length is 19 due to the starting value is 2), with the length of 50 which is

$$\tilde{\eta}([2;\, 50]),\ \tilde{\eta}([50 \cdot (k-1) + 1;\, 50 \cdot k]) \text{ for } k = \overline{2, 20} \quad (33)$$

(the very first interval length is 49 by the same reason), and with the length of 100 which is

$$\tilde{\eta}([2;\, 100]),\ \tilde{\eta}([100 \cdot (k-1) + 1;\, 100 \cdot k]) \text{ for } k = \overline{2, 10} \quad (34)$$

(the very first interval length is 99). Fig. 4 shows these three versions of the sliding average of the relative difference in Fig. 1, wherein the wider window is marked with a thicker line of a lighter color. Generally speaking, each of sliding averages (32)—(34) seems resembling the exponentially-like decreasing relative difference and its running average by (27). However, the slight advantage (of about 0.5 % to 1.8 %) of the descending job order input quickly vanishes since 400 jobs. Moreover, the advantage, if any, is too slim in scheduling between 400 and 1000 jobs. Fig. 5 showing the three versions of the sliding average of the relative difference in Fig. 2 confirms these doubts: in scheduling between 200 and 800 jobs, the ascending job order input is faster. Finally, the three versions of the sliding average of the relative difference in Fig. 3 shown in Fig. 6 appear disorderly also.

Although it seems that some advantage of the descending job order input exists in scheduling up to 100 jobs (owing to Fig. 4—6), its statistical reliability is not high. Again, these figures reveal that the advantage, if any, is too unstable and unpredictable. As the sliding window becomes wider, the sliding averages over the respective intervals appear more "pressed" to the horizontal zero level line.

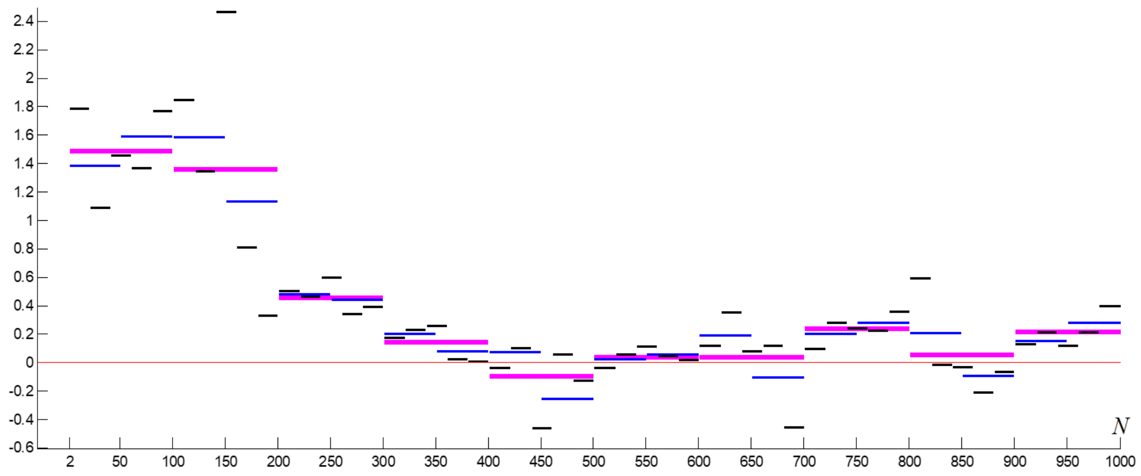

Fig. 4. The three versions of the sliding average of the relative difference in Fig. 1: — – Sliding window 20; — – Sliding window 50; — – Sliding window 100

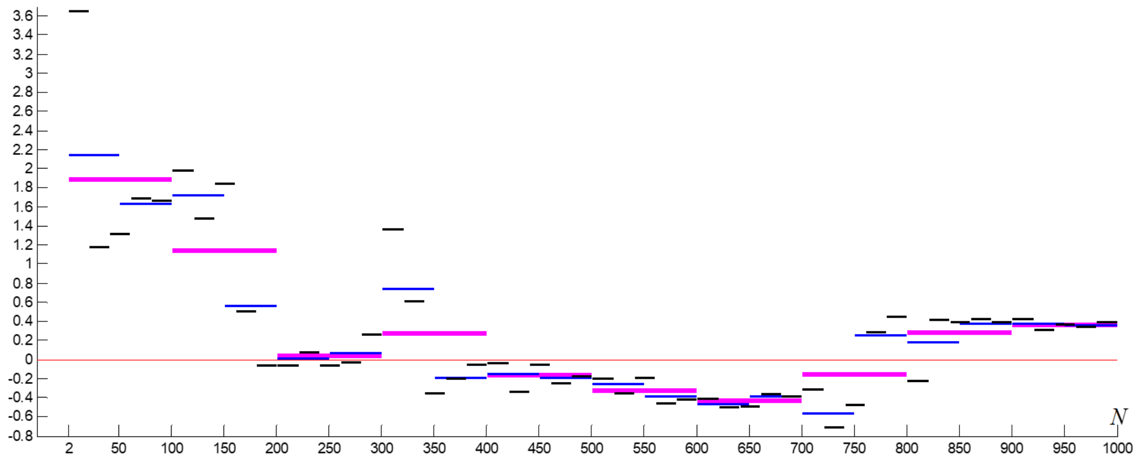

Fig. 5. The three versions of the sliding average of the relative difference in Fig. 2: — – Sliding window 20; — – Sliding window 50; — – Sliding window 100

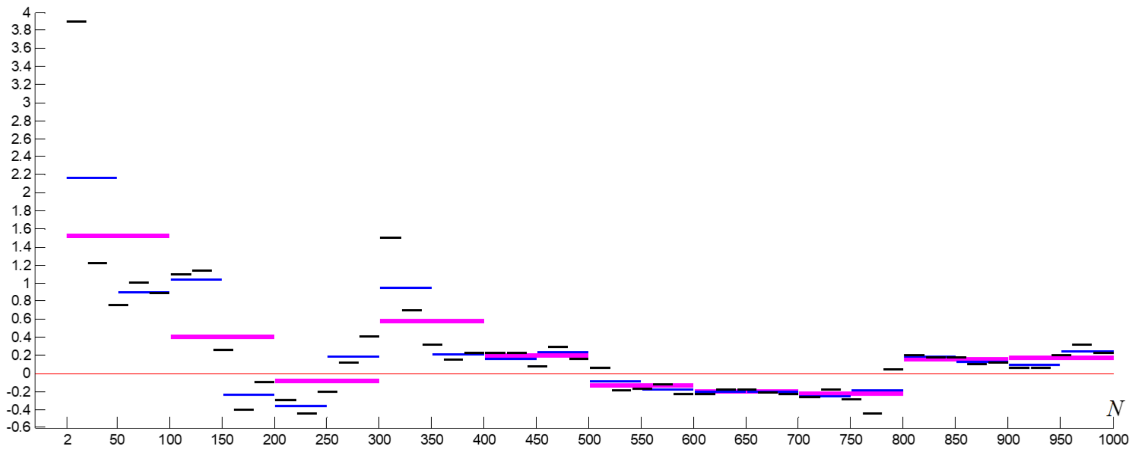

Fig. 6. The three versions of the sliding average of the relative difference in Fig. 3: — – Sliding window 20; — – Sliding window 50; — – Sliding window 100

Discussion

As the heuristic is extremely fast itself (1000 jobs are scheduled within 150 ms to 170 ms on the computational study's equipment), the real-time difference between the computation times by ascending order and descending order is too tiny. This is why the relative difference between the computation times is so unstable. Another aspect of the heuristic is susceptibility of its algorithm to the processor on which it is executed (the plots in Fig. 1–6 will likely appear differently when using other operating system and processor). Eventually, the seeming advantage of the descending job order input in scheduling up to 100 jobs is about 0.4 % to 1.5 %, by scheduling 100 jobs within 5.3 ms to 8.6 ms, which can be converted into a real benefit only if a long series of such scheduling problems are solved (on the same equipment). For example, solving a series of 10000 problems with 100 jobs (where each problem is solved by about 8 ms) by the descending job order input relying on its 1.5 % advantage saves just 1.2 seconds. So, the efficiency of either job order input can be practically treated equal.

Conclusions

By using the heuristic for the case of tight-tardy progressive idling-free 1-machine preemptive scheduling with job priority weights, the significance of the job order input is much lower than that for the case of jobs without priorities. With assigning the job priority weights, the job order input becomes further "dithered", adding randomly scattered priority weights to randomly scattered job lengths and partially randomized due dates. On average, the descending job order input is believed to give a tiny advantage in computation time in scheduling up to 100 jobs. However, this advantage, if any (being tinier than that in the case of random job lengths without priorities), quickly vanishes as the number of jobs increases.

As a final result, it is better to compose job scheduling problems which would be closer to the case with equal-length jobs without priorities, where the saved computational time can be counted in hours. Even if the job lengths and priority weights are scattered, it is recommended to artificially (manually) "flatten" them, if possible, and then select the corresponding job order input for the heuristic using the description in [7]. When artificial manipulations over job processing periods and job priority weights are impossible, it is recommended to use the descending job order input in scheduling up to 100 jobs, and either job order input in scheduling more than 100 jobs, although substantial benefits are not expected in this case.

References

[1] V.V. Romanuke, "Efficient exact minimization of total tardiness in tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs", *KPI Sci. News*, no. 1, pp. 27–39, 2020. doi: 10.20535/kpi-sn.2020.1.180877

[2] R. Panneerselvam, "Simple heuristic to minimize total tardiness in a single machine scheduling problem", *Int. J. Adv. Manufact. Technol.*, vol. 30, no. 7–8, pp. 722–726, 2006. doi: 10.1007/s00170-005-0102-1

[3] M. Batsyn *et al.*, "Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time", *Optimiz. Method. & Software*, vol. 29, no. 5, pp. 955–963, 2014.
doi: 10.1080/10556788.2013.854360

[4] D. Rupanetti and H. Salamy, "Task allocation, migration and scheduling for energy-efficient real-time multiprocessor architectures", *J. Syst. Architec.*, vol. 98, pp. 17–26, 2019. doi: 10.1016/j.sysarc.2019.06.003

[5] V.V. Romanuke, "Tight-tardy progressive idling-free 1-machine preemptive scheduling by heuristic's efficient job order input", *KPI Sci. News*, no. 3, pp. 32–42, 2020. doi: 10.20535/kpi-sn.2020.3.199850

[6] F. Jaramillo and M. Erkoc, "Minimizing total weighted tardiness and overtime costs for single machine preemptive scheduling", *Comp. Industr. Eng.*, vol. 107, pp. 109–119, 2017. doi: 10.1016/j.cie.2017.03.012

[7] V.V. Romanuke, "Heuristic's job order efficiency in tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs", *KPI Sci. News*, no. 2, pp. 64–73, 2020. doi: 10.20535/kpi-sn.2020.2.181869

[8] M.L. Pinedo, *Planning and Scheduling in Manufacturing and Services.* New York: Springer-Verlag, 2009, 536 p.
doi: 10.1007/978-1-4419-0910-7

[9] V.V. Romanuke, "Job order input for efficient exact minimization of total tardiness in tight-tardy progressive single machine scheduling with idling-free preemptions", *Scientific Papers O. S. Popov Odesa Nat. Academy Telecommun.*, no. 1, pp. 19–36, 2020. doi: 10.33243/2518-7139-2020-1-1-19-36

[10] V.V. Romanuke, "Total weighted tardiness exact minimization by efficiently inputting jobs to tight-tardy progressive single machine scheduling with idling-free preemptions", *Digital Technol.*, no. 27, pp. 41–55, 2020..

[11] V.V. Romanuke, “Minimal total weighted tardiness in tight-tardy single machine preemptive idling-free scheduling”, *Appl. Comp. Syst.*, vol. 24, no. 2, pp. 150–160, 2019. doi: 10.2478/acss-2019-0019

[12] M. Tamannaei and M. Rasti-Barzoki, “Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem”, *Comp. Industr. Eng.*, vol. 127, pp. 643–656, 2019. doi: 10.1016/j.cie.2018.11.003

[13] R.T. Kneusel, *Random Numbers and Computers*. Springer International Publishing, 2018, 260 p. doi: 10.1007/978-3-319-77697-2

[14] S. Vigna, “On the probability of overlap of random subsequences of pseudorandom number generators”, *Inform. Process. Lett.*, vol. 158, ID 105939, 2020. doi: 10.1016/j.ipl.2020.105939

[15] M.H. Ardakani *et al.*, “Sliding dynamic data window: Improving properties of the incremental learning methods”, *Comput. Aided Chemical Eng.*, vol. 40, pp. 1663–1668, 2017. doi: 10.1016/B978-0-444-63965-3.50279-8

[16] N. Gasmi *et al.*, “Chapter 21 - Nonlinear filtering design for discrete-time systems using sliding window of delayed measurements”, in *Stability, Control and Applications of Time-delay Systems*. Butterworth-Heinemann, 2019, pp. 423–439. doi: 10.1016/B978-0-12-814928-7.00021-4

В.В. Романюк

ЩІЛЬНЕ ПРОГРЕСУЮЧЕ 1-МАШИННЕ ПЛАНУВАННЯ З ПЕРЕМИКАННЯМИ БЕЗ ПРОСТОЮ З ВАГАМИ ПРІОРИТЕТУ ЗАВДАНЬ ЗА ЕФЕКТИВНОГО ПОРЯДКУ ВВОДУ ЗАВДАНЬ У ЕВРИСТИЦІ

Проблематика. У постановці задачі мінімізації загального зваженого запізнювання за евристикою на основі використання залишкового наявного ресурсу та залишкового періоду до обробки існують два протилежних способи вводу даних: дати запуску завдань задаються порядком зростання чи спадання. Нещодавно було встановлено, що планування декількох рівноцінних завдань є очікувано швидшим за висхідного порядку і водночас планування від 30 до 70 рівноцінних завдань на 1,5–2,5 % швидше за спадного порядку. Для кількості рівноцінних завдань між приблизно 90 і 250 висхідний порядок знову призводить до скорочення часу обчислень. У випадку, коли завдання мають різні об’єми, істотність порядку завдань є значно нижчою. У середньому спадний порядок вводу завдань дає крихітну перевагу в часі обчислень. Ця перевага спадає зі зростанням кількості завдань.

Мета дослідження. Встановити, чи порядок завдань є істотним у складанні розкладів за допомогою евристики для випадку, коли завдання мають різні об’єми з вагами їх пріоритетів. Ефективність порядку завдань буде досліджено на прикладі щільного прогресуючого 1-машинного планування з перемиканнями без простою.

Методика реалізації. Проводиться обчислювальне дослідження з метою оцінки усередненого часу обчислення як для висхідного порядку, так і для спадного порядку дат запуску завдань. Спочатку оцінюється час обчислень для послідовності задач складання розкладів за висхідного порядку вводу завдань. Далі в кожному екземплярі цієї послідовності об’єми завдань, ваги пріоритетів, дати запуску та дати прийому виконання обертаються, утворюючи відповідний екземпляр для спадного порядку вводу завдань, для якого також оцінюється час обчислення.

Результати дослідження. Значимість порядку вводу завдань є значно нижчою, ніж у випадку завдань без пріоритетів. За приписування завданням ваг пріоритетів порядок вводу завдань стає надалі більш “розмитим” із додаванням випадково розкиданих ваг пріоритетів до випадково розкиданих об’ємів завдань і частково рандомізованих дат прийому виконання. Передбачається, що в середньому спадний порядок вводу завдань дає крихітну перевагу в часі обчислень при складанні розкладів до 100 завдань. Однак ця перевага (котра є ще більш крихітною, ніж у випадку з рандомізованими об’ємами завдань без пріоритетів), якщо все ж існує, швидко зникає зі збільшенням кількості завдань.

Висновки. Доведено, що кращим варіантом є складання таких задач планування завдань, які були б ближчими до випадку з рівноцінними завданнями без пріоритетів, де збережений обчислювальний час може нараховувати години. Навіть якщо об’єми завдань і ваги пріоритетів мають розкид, рекомендовано штучно “розгладжувати” їх. Коли ж штучні маніпуляції з періодами обробки завдань і вагами пріоритетів завдань неможливі, рекомендовано використовувати спадний порядок вводу завдань у плануванні до 100 завдань, а також будь-який порядок вводу завдань у плануванні понад 100 завдань, хоча істотні корисності у цьому випадку не очікуються.

Ключові слова: планування завдань на одній машині з перемиканнями; загальне зважене запізнювання; евристика; висхідний порядок завдань; спадний порядок завдань; час обчислень; ефективний порядок завдань.

В.В. Романюк

ПЛОТНОЕ ПРОГРЕССИРУЮЩЕЕ 1-МАШИННОЕ ПЛАНИРОВАНИЕ С ПЕРЕКЛЮЧЕНИЯМИ БЕЗ ПРОСТОЯ С ВЕСАМИ ПРИОРИТЕТА ЗАДАНИЙ ПРИ ЭФФЕКТИВНОМ ПОРЯДКЕ ВВЕДЕНИЯ ЗАДАНИЙ В ЭВРИСТИКЕ

Проблематика. В постановке задачи минимизации общего взвешенного запаздывания по эвристике на основе использования остаточного имеющегося ресурса и остаточного периода к обработке существуют два противоположных способа ввода данных: даты запуска заданий задаются в порядке возрастания или убывания. Недавно было установлено, что планирование нескольких равноценных заданий ожидаемо быстрее при восходящем порядке, тогда как планирование от 30 до 70 равноценных заданий на 1,5–2,5 % быстрее при нисходящем порядке. Для количества равноценных заданий между примерно 90 и 250 восходящий порядок снова приводит к сокращению времени вычислений. В случае, когда задания имеют различные объёмы, значимость порядка заданий значительно понижается. В среднем нисходящий порядок введения заданий даёт крошечный перевес во времени вычислений. Этот перевес убывает с ростом числа заданий.

Цель исследования. Установить, является ли порядок заданий значимым в составлении расписаний с помощью эвристики для случая, когда задания имеют различные объёмы с весами их приоритетов. Эффективность порядка заданий будет исследовано на примере плотного прогрессирующего 1-машинного планирования с переключениями без простоя.

Методика реализации. Проводится вычислительное исследование с целью оценки усреднённого времени вычисления как для восходящего порядка, так и для нисходящего порядка дат запуска заданий. Сначала оценивается время вычислений для последовательности задач составления расписаний при восходящем порядке введения заданий. Затем в каждом экземпляре этой последовательности объёмы заданий, веса приоритетов, даты запуска и даты приёма выполнения оборачиваются, образуя таким образом соответствующий экземпляр для нисходящего порядка введения заданий, для которого также оценивается время вычисления.

Результаты исследования. Значимость порядка введения заданий значительно ниже, чем в случае заданий без приоритетов. С приписыванием заданиям весов приоритетов порядок введения заданий становиться ещё более “размытым” с прибавлением случайно разбросанных весов приоритетов к случайно разбросанным объёмам заданий и частично рандомизированным датам приёма выполнения. Предполагается, что в среднем нисходящий порядок введения заданий даёт крошечное преимущество во времени вычислений при составлении расписаний до 100 заданий. Однако это преимущество (которое является ещё более крошечным, чем в случае с рандомизированными объёмами заданий без приоритетов), если даже и существует, быстро исчезает с увеличением количества заданий.

Выводы. Доказано, что лучшим вариантом является составление таких задач планирования заданий, которые были бы ближе к случаю с равноценными заданиями без приоритетов, где сэкономленное вычислительное время может насчитывать часы. Даже если объёмы заданий и веса приоритетов имеют разброс, рекомендуется искусственно “разглаживать” их. Когда же искусственные манипуляции с периодами обработки заданий и весами приоритетов заданий невозможны, рекомендуется использовать нисходящий порядок введения заданий в планировании до 100 заданий, а также любой порядок введения заданий в планировании более чем 100 заданий, хотя существенные полезности в этом случае не ожидаются.

Ключевые слова: планирование заданий на одной машине с переключениями; общее взвешенное запаздывание; эвристика; восходящий порядок заданий; нисходящий порядок заданий; время вычислений; эффективный порядок заданий.

Рекомендована Радою
факультету прикладної математики
КПІ ім. Ігоря Сікорського

Надійшла до редакції
31 липня 2020 року

Прийнята до публікації
10 грудня 2020року