DOI: 10.20535/kpi-sn.2020.3.199850
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 BY HEURISTIC'S EFFICIENT JOB ORDER INPUT

Background. In setting a problem of minimizing total 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.
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. 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. Instances of the job scheduling problem are generated so that schedules which can be obtained trivially, without the heuristic, are excluded.
Results. 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 as it was for the case of equal-length jobs.
Conclusions. The significance of the job order input is much lower than that for the case of equal-length jobs. 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.

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

Introduction

The exact minimization of total tardiness is possible just for a few jobs whose processing periods are not very long [1, 2]. Heuristics are the only means which capable of scheduling hundreds and thousands of jobs [3, 4]. Moreover, the entire schedule can be an extremely long sequence of jobs [5], 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). The heuristic based on the remaining available period and remaining processing period [3] is closely the best one. Article [6] 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 [7]. Article [6] 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 [6] 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.

The case when the jobs have different lengths (i. e., whose number of processing periods varies) is an extension (continuation) of the case studied in [6]. Now, the influence of varying job lengths 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. The four following tasks will be fulfilled for achieving this goal. Firstly, the heuristic is shortly stated for the case of when the jobs have different lengths. Then, secondly, generation of the job scheduling problem instances is stated for this case. Thirdly, a computational study is to be 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 heuristic's efficient job order input exists.

The heuristic based on remaining available and processing periods

The problem of total tardiness minimization, considering job processing periods (job lengths), release dates and due dates, is intentionally simplified to that all the parameters are given as natural numbers. 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 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 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, 6]). In particular, the release dates can be given in ascending order as follows:

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

This is the common way of inputting the data into the algorithm of solving the problem. Strictly speaking, the release dates can be permuted as one likes or needs to satisfy some external conditions (obviously, the job processing periods 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

$$H_n \leq H_{n+1} \quad \text{and} \quad d_n \leq d_{n+1} \quad \forall n = \overline{1,\ N-1} \tag{7}$$

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

$$H_n \geq H_{n+1} \quad \text{and} \quad d_n \geq d_{n+1} \quad \forall n = \overline{1,\ N-1} \tag{9}$$

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

The heuristic based on remaining available and processing periods builds stepwise a schedule $\tilde{\mathbf{S}} = [\tilde{s}_t]_{1\times T}$, where $T = \sum_{n=1}^{N} H_n$, as time t progresses. Before the start,

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

Then, for every set of available jobs

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

the remaining available period is

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

and a subset

$$A^*(t) = \arg\max_{i\in A(t)}(\max\{q_i,\ b_i\})^{-1} \tag{14}$$

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

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

then

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

otherwise

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

whence

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

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 [6]. Thus, job n is completed after moment $\tilde{\theta}(n;\ H_n)$ if

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

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

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

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

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

Generation of the job scheduling problem instances

As the jobs can have different lengths, they should be randomly generated. Let a method suggested in [9] be used for this:

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

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 [6]. When job lengths (19) 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 $\{H_n\}_{n=1}^{N}$, release dates (3), and due dates (8) is computed by the heuristic with statements (11)–(18). Alternatively, a descending order schedule by job lengths

$$\{H_n\}_{n=1}^{N} \quad \text{after} \quad H_j^{(\mathrm{obs})} = H_j \quad \forall\, j = \overline{1,\ N}$$

$$\text{and} \quad H_n = H_{N-n+1}^{(\mathrm{obs})} \quad \text{for} \quad n = \overline{1,\ N}\,, \tag{20}$$

release dates (4), and due dates (10) is computed as well. In fact, job lengths (20), 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 $\{H_n\}_{n=1}^{N}$, release dates (3), and due dates (8) for the ascending job order input. However, the version with generating the ascending job order input and descending job order input separately, independently of each other, is to be studied also.

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

$$\tau_{Asc}(N) = \frac{1}{C}\sum_{c=1}^{C} \tau_{Asc}(N,\ c) \tag{21}$$

and

$$\tau_{Desc}(N) = \frac{1}{C}\sum_{c=1}^{C} \tau_{Desc}(N,\ c)\,. \tag{22}$$

In percentage terms, the relative difference between computation times (21) and (22) is

$$\eta(N) = 100 \cdot \frac{\tau_{Asc}(N) - \tau_{Desc}(N)}{\tau_{Asc}(N)}. \qquad (23)$$

For a few hundred instances at a fixed number of jobs, the variance of relative difference (23) is roughly the same as for one hundred instances. This is why relative difference (23) will be estimated by $N = \overline{2, 1000}$ for $C = 100$ (generating more instances will not make the estimation more effective).

Computational study

Just as in article [6], the computational study is executed on CPU Intel Core i5-7200U@2.50 GHz using MATLAB R2018a. First of all, the version with generating the ascending job order input and descending job order input separately, independently of each other, is studied. Here ascending and descending job order inputs are generated by (2) and (19) using two different randomizers. Relative difference (23) is shown in Fig. 1 for this version whose average relative difference

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.4052 \qquad (24)$$

indicates a little advantage of the descending job order input.

Nevertheless, it is hard to assess the advantage with only relative difference (23), even having the horizontal zero level line put on the plot. This is why running average relative difference

$$\tilde{\eta}(N) = \frac{1}{N-1} \cdot \sum_{m=2}^{N} \eta(m) \quad \text{for} \quad N = \overline{2, 1000} \qquad (25)$$

should be additionally used for the analysis. Running average relative difference (25) is shown in Fig. 2, where average (24) is a partial case (the ultimate single estimation point) of averages (25). Not paying attention at fluctuations caused by computational microartifacts, it is well seen that the descending job order input is really faster than the ascending job order input, although its advantage decreases as the number of jobs increases. If up to 200 jobs are scheduled, the average relative difference is about 2 % to 3 %.

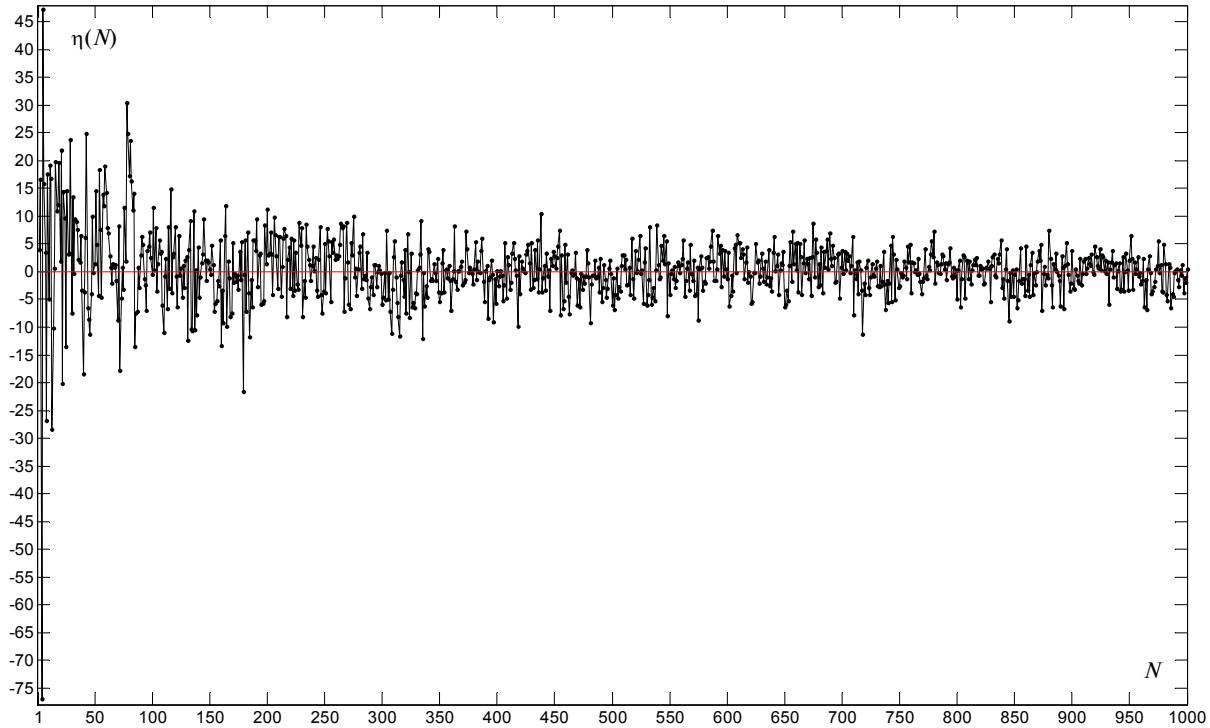

Fig. 1. Relative difference (23) for the version where the ascending job order input and descending job order input are generated separately (independently of each other) using different randomizers

The comparison of two different randomizers for the ascending and descending job order inputs resulted in Fig. 1 with the relative difference and Fig. 2 with the running average relative difference is not purely correct. The matter is the correct comparison is to be on the single randomizer for generating random due date shifts (2) and random job lengths (19), where job lengths, release dates, and due dates for the descending job order input are obtained by just reversing those ones for the ascending job order input. Relative difference (23) for this version is shown in Fig. 3. A refined version of the plot in Fig. 3 is presented in Fig. 4 using the same presentation style, by only cutting values of the relative difference to the range of $-5\,\%$ to $5\,\%$. Now, the appearance of the relative difference differs from that of Fig. 1: it fluctuates less but has more visible computational artifacts compared to Fig. 1. The drop between approximately 560 and 600 jobs seems to be a complex computational artifact (something similar is seen nearby 300 jobs). Nevertheless, it is clearly seen in both Figs. 3 and 4 that the descending job order input is faster in scheduling up to 400 jobs. This does not contradict the running average relative difference in Fig. 2.

The average relative difference

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.4755 \qquad (26)$$

for this version of the (first) mutual randomizer is slightly greater than that (24) for two different randomizers. Meanwhile, the running average relative difference for the first mutual randomizer version appears as a much smother curve resembling more an exponential decrease (Fig. 5). Fig. 5 proves that the descending job order input is faster, although its advantage smoothly decreases down to average relative difference (26). Unlike the running average relative difference for the version with different randomizers in Fig. 2, the running average relative difference for the first mutual randomizer version does not have computational artifacts. However, one should remember that the running average relative difference in Fig. 5 has "eaten" the huge computational artifacts in Fig. 3.

How badly will Fig. 3 or Fig. 4 change if to run the last computational experiment once again but with another randomizer (randomizers differ in just an initial state for generating pseudorandom num-

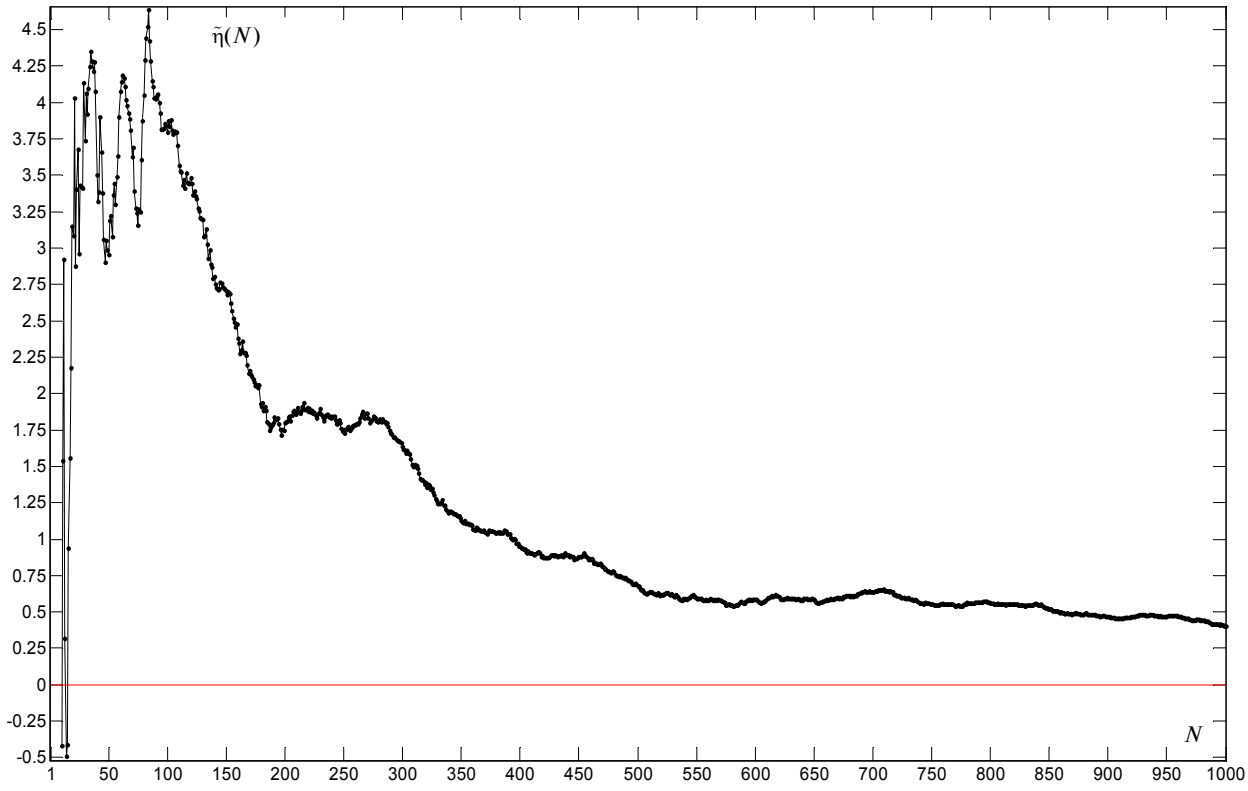

Fig. 2. Running average relative difference (25) for the version with different randomizers for the ascending and descending job order inputs in Fig. 1

bers)? The answer to this question is particularly shown in Fig. 6 resembling Fig. 3 in computational artifacts. Another complex computational artifact is seen as a drop between approximately 390 and 460 jobs (Fig. 7) being similar to that in Fig. 4 between approximately 560 and 600 jobs. Unlike Figs. 3 and 4, showing that the descending job order input is faster in scheduling up to 400 jobs, Figs. 6 and 7 "claim"

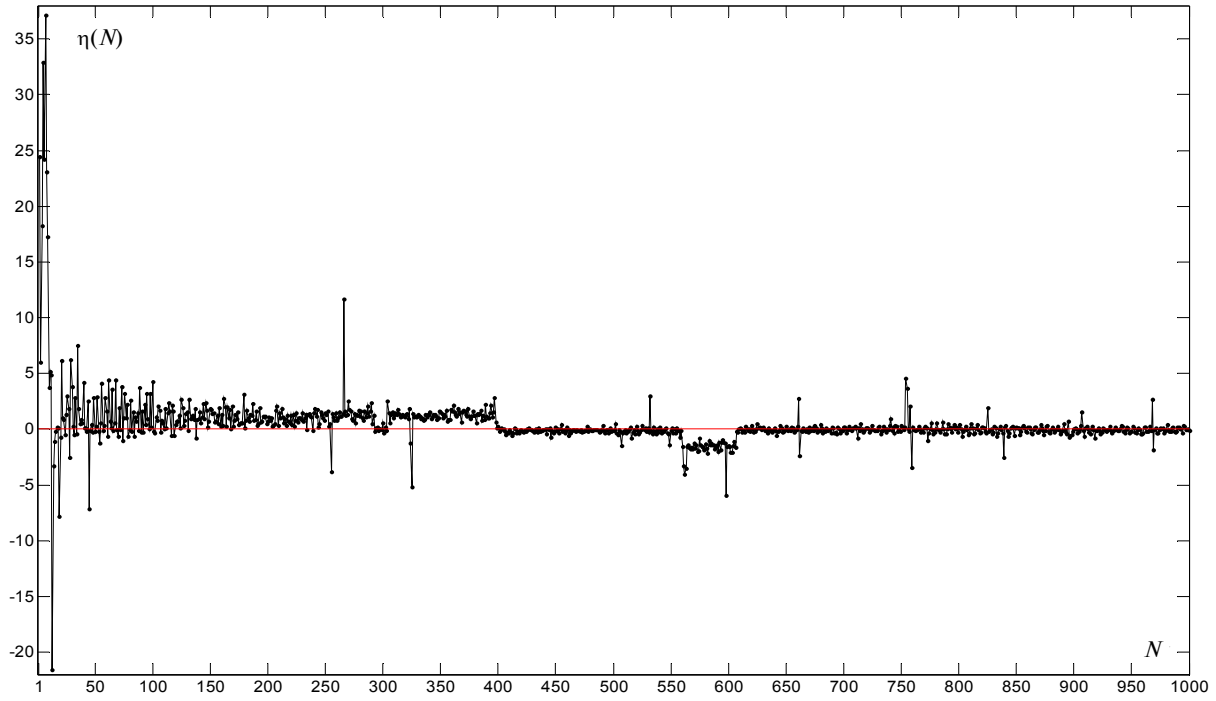

Fig. 3. Relative difference (23) for the version where job lengths, release dates, and due dates for the descending job order input are obtained by just reversing those ones for the ascending job order input

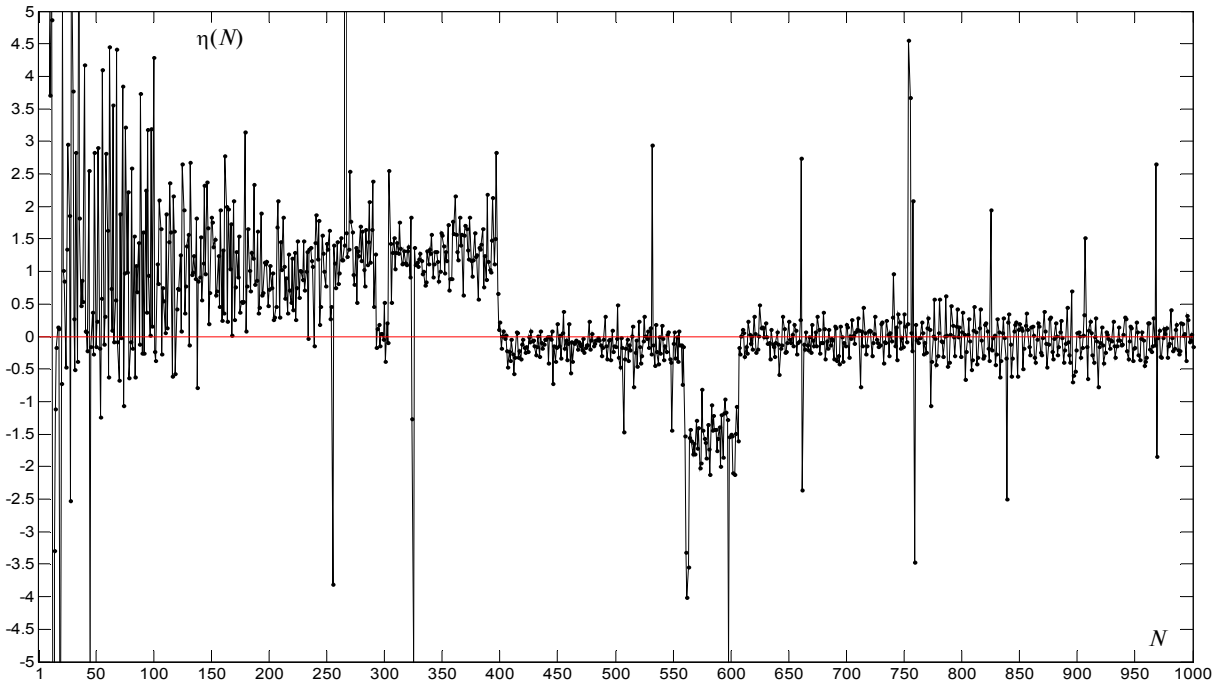

Fig. 4. Relative difference (23) in Fig. 3 (the first mutual randomizer) cut to the range of −5 % to 5 %

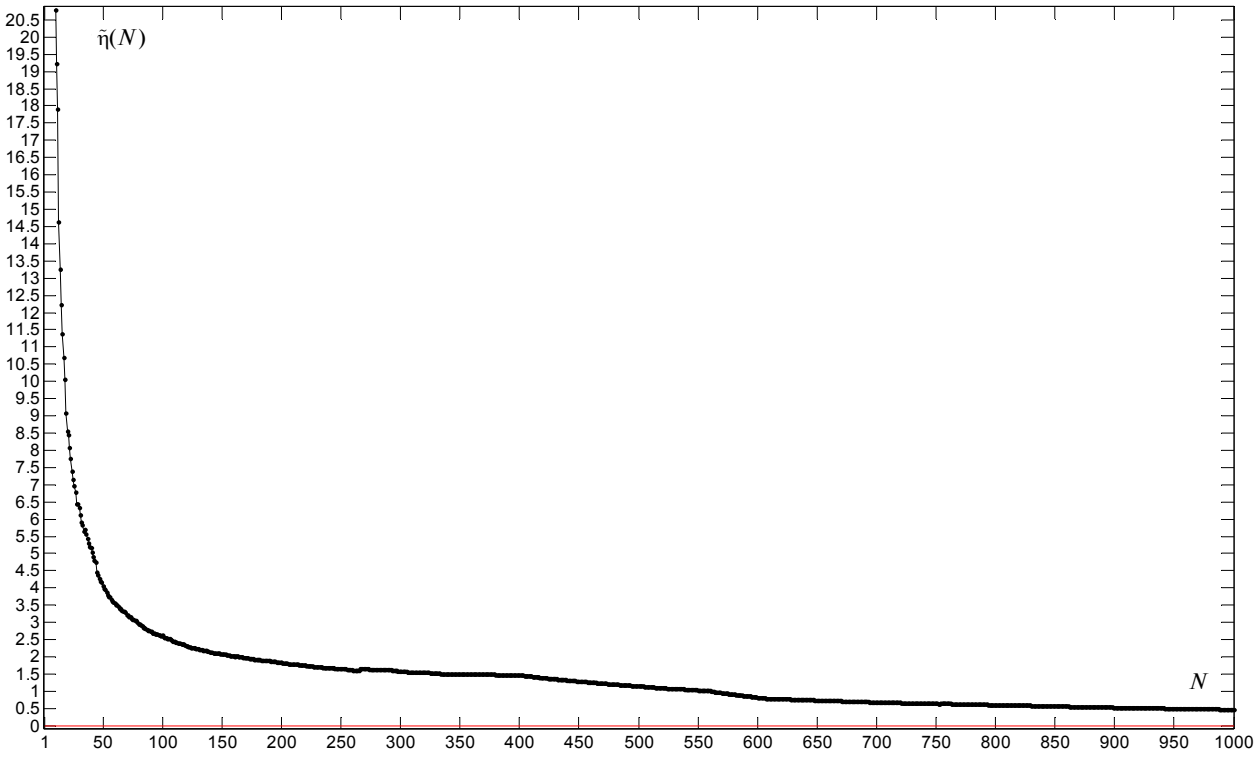

Fig. 5. Running average relative difference (25) for the first mutual randomizer version (Fig. 3 and Fig. 4 with the cut relative difference)

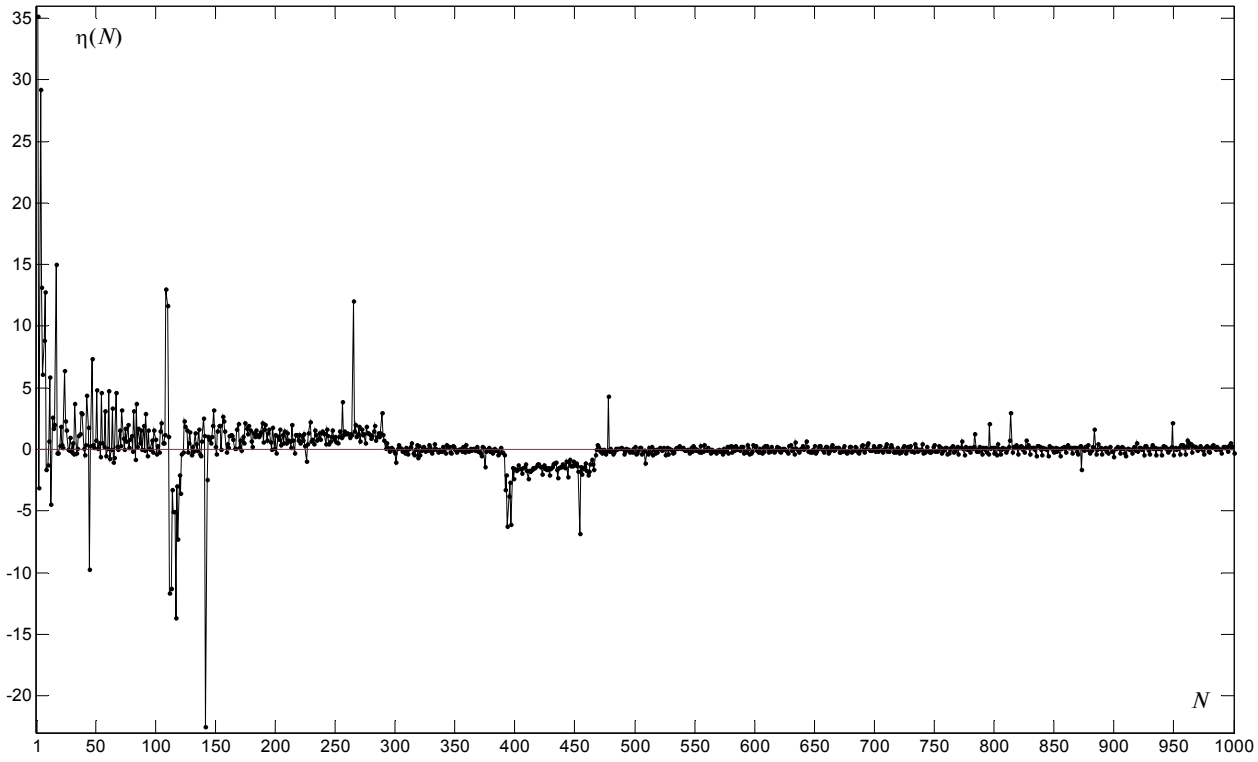

Fig. 6. Relative difference (23) for the second version of the mutual randomizer (job lengths, release dates, and due dates for the descending job order input are obtained by just reversing those ones for the ascending job order input)

that the descending job order input is faster in scheduling up to 300 jobs, whereas the interval from 300 to 400 jobs seems to have no preference to either the ascending or descending job order input.

However, the running average relative difference for the second mutual randomizer version (Fig. 8) proves that the descending job order input is faster anyway. It is not as smooth as that in Fig. 5,

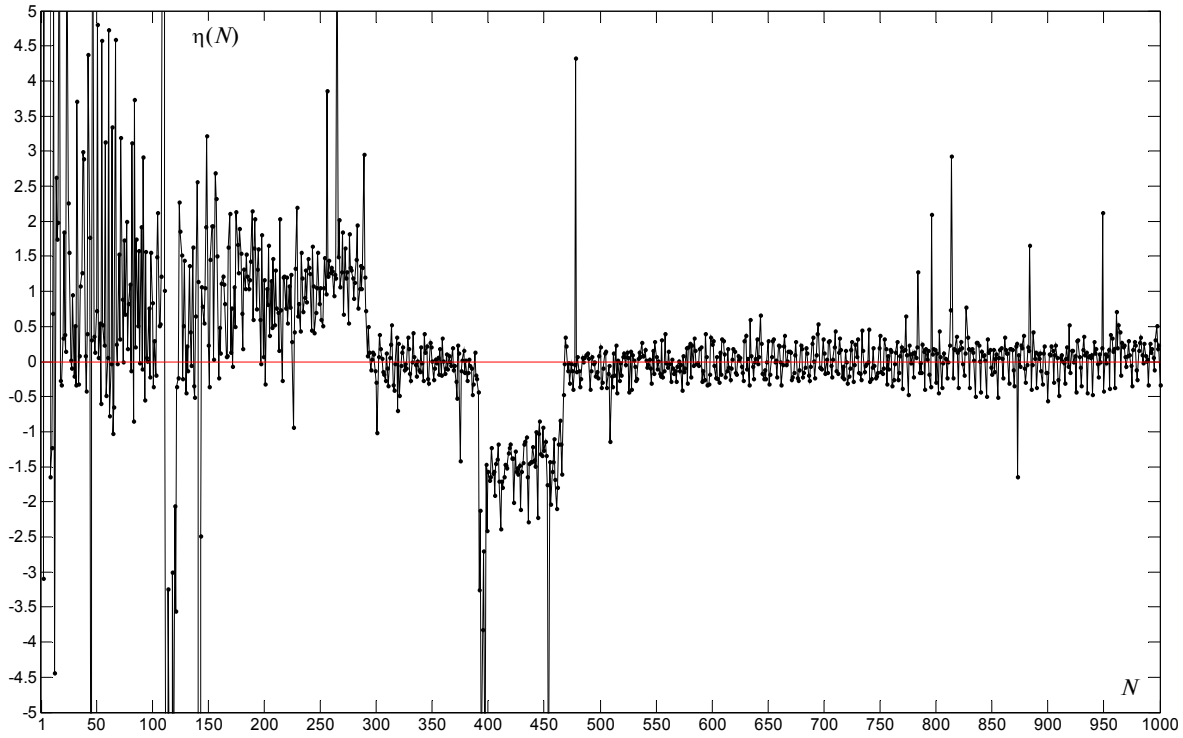

Fig. 7. Relative difference (23) in Fig. 6 (the second mutual randomizer) cut to the range of −5 % to 5 %

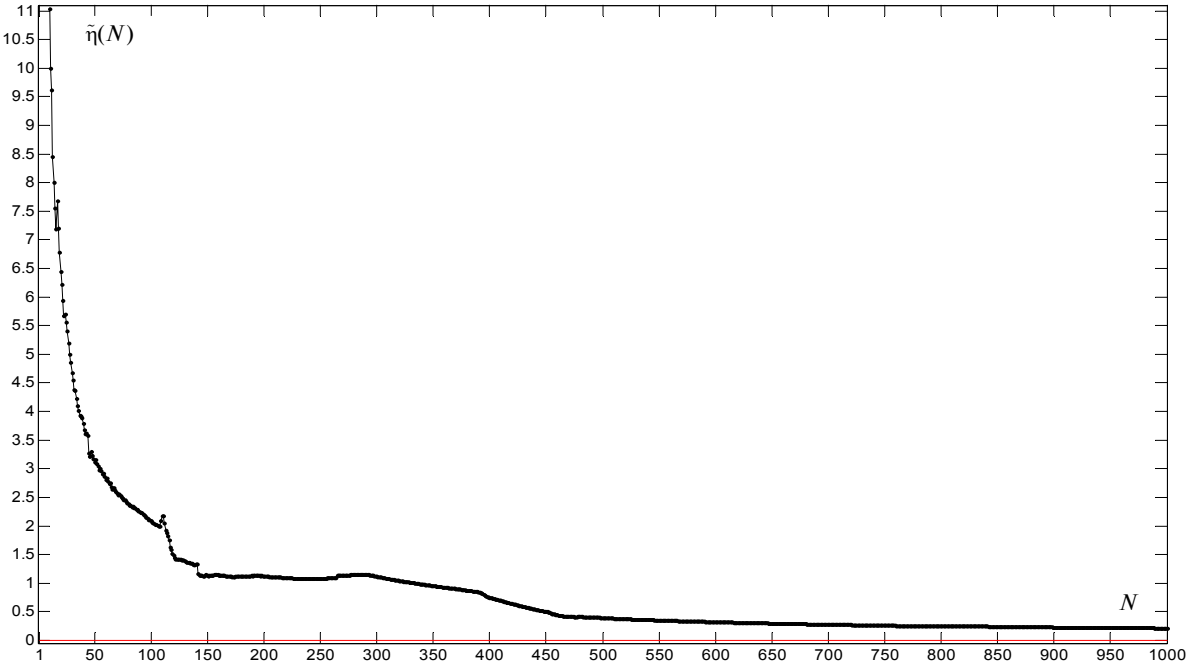

Fig. 8. Running average relative difference (25) for the second mutual randomizer version (Figs. 6 and 7 with the cut relative difference)

and even some computational artifacts are seen till approximately 500 jobs (remember that the running average relative difference in Fig. 8 has "eaten" the huge computational artifacts in Fig. 6), but it still resembles more an exponential decrease down to the average relative difference

$$\tilde{\eta}(1000) = \frac{1}{999} \cdot \sum_{m=2}^{1000} \eta(m) \approx 0.2154 \qquad (27)$$

being roughly twice as lesser as average relative differences (24) and (26). Here the advantage of the descending job order input can be thought of as a lower bound of its estimation.

It is necessary to note that the heuristic is extremely fast itself. Thus, 1000 jobs whose lengths are within range of 2 to 17 are scheduled in about slightly more than 150 ms, whereas 500 such jobs are scheduled in about as thrice as faster. Scheduling 100 jobs is completed in less than 6 ms. So, the real-time difference between the computation times by ascending order and descending order in scheduling up to 100 jobs consisting of 2 to 17 parts each is pretty tiny. This difference may become more impressive if job lengths are increased.

Discussion

Another way to see the real-time difference between the computation times is to work on a long series of smaller job scheduling problems (which factually have been studied). However, the factual speed-up will be far from very significant. Thus, working on a series of 50000 problems of scheduling 200 jobs whose lengths are within range of 2 to 17 is executed by the ascending job order input in 650.603 seconds. In its turn, such a series is executed in 649.609 seconds by the descending job order input (just a 0.153 % advantage), so almost 1 second is saved (which may be still significant for many computational systems). A much greater percentage of the computation speed advantage can be obtained in scheduling a lesser number of jobs: e. g., working on a series of 50000 problems of scheduling 10 jobs whose lengths are within range of 2 to 14 is executed by the ascending job order input in 15.1 seconds. In its turn, such a series is executed in 14.861 seconds by the descending job order input (a 1.5824 % advantage), but it saves only 0.239 seconds. Surely, not every instance will be scheduled faster by the descending job order input. The matter is the job order is defined, apart from the release dates, by the due dates. The latter are formed by adding random shifts (2), so the factual order of the due dates can be estimated only by their trend line. If due dates (by the descending job order input) are badly scattered around their trend line, the descending job order input will probably have a weak speed-up effect or no speed-up effect at all. For instance, Fig. 9 shows the due dates for the mentioned example with 200 jobs, where the trend is easily seen. Contrary to that, the due dates for the example with 10 jobs shown in Fig. 10 are badly scattered, and it is almost impossible to determine to which order of the release dates they correspond.

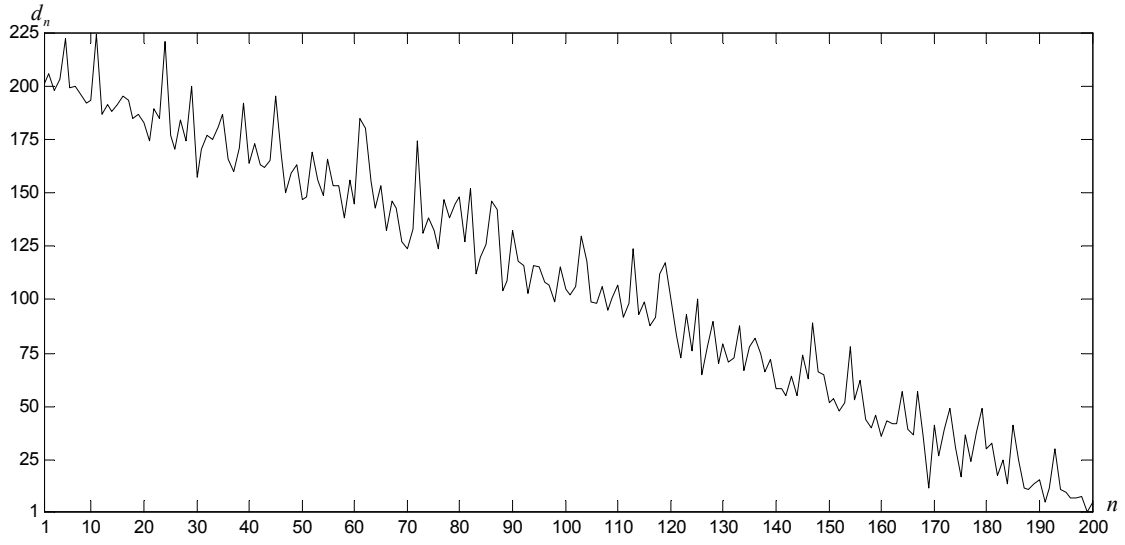

Fig. 9. The due dates corresponding to the release dates in descending order for an example of scheduling 200 jobs whose lengths are within range of 2 to 17

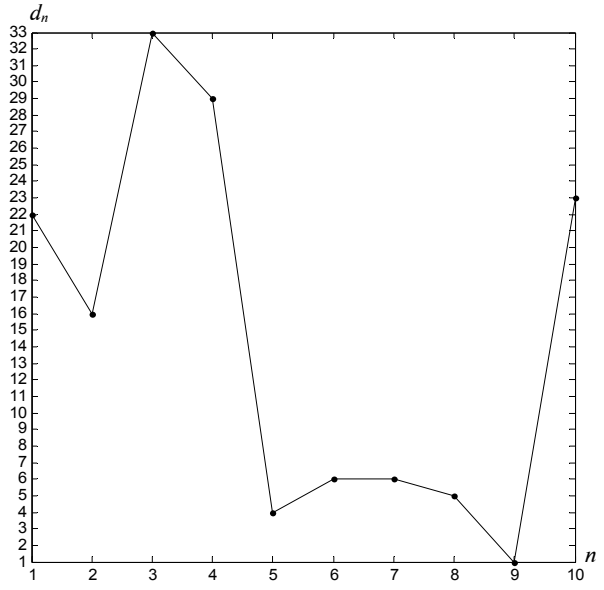

Fig. 10. The due dates corresponding to the release dates in descending order for an example of scheduling 10 jobs whose lengths are within range of 2 to 14

Eventually, the advantage of the descending job order input, whose lower bound can be roughly estimated as that in Fig. 8, should be perceived only as on average. Therefore, the descending job order input does not guarantee that a set of jobs will be scheduled faster, even by the least registered average relative difference (27). Moreover, unlike the case with scheduling equal-length jobs, scheduling greater amounts of jobs (say, a few thousand jobs whose lengths are within range of 2 to 17) does not lead to saving considerable computational time.

Conclusions

By using the heuristic for the case of tight-tardy progressive idling-free 1-machine preemptive scheduling, 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 as it was for the case of equal-length jobs. 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 (see Fig. 7).

The job order input research should be furthered by studying the case when different priority weights are considered for minimizing total weighted tardiness by the heuristic (based on remaining available and processing periods). Then the final comparison of the respective results for the three classes of job scheduling problems (total tardiness by equal job lengths [6], total tardiness by different job lengths, total weighted tardiness) should be made. These results are to be arranged and the corresponding recommendations are to be formulated.

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] W.-Y. Ku and J. C. Beck, "Mixed Integer Programming models for job shop scheduling: A computational analysis", *Comp. Oper. Res.*, vol. 73, pp. 165–173, 2016. doi: 10.1016/j.cor.2016.04.006

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

[4] R. Panneerselvam, "Simple heuristic to minimize total tardiness in a single machine scheduling problem", *Int. J. Adv. Manufact. Technol.*, vol. 30, iss. 7-8, pp. 722–726, 2006. doi: 10.1007/s00170-005-0102-1

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

[6] 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

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

[8] 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 of O.S. Popov Odesa National Academy of Telecommunications*, no. 1, pp. 19–36, 2020.

[9] 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

В.В. Романюк

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

Проблематика. У постановці задачі мінімізації загального запізнювання за евристикою на основі використання залишкового наявного ресурсу та залишкового періоду до обробки існують два протилежних способи вводу даних: дати запуску завдань задаються в порядку зростання або спадання. Нещодавно було встановлено, що планування декількох рівноцінних завдань є очікувано швидшим за висхідного порядку, тоді як 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 приводить до скорочення часу обчислень.

Мета дослідження. Метою є встановлення того, чи порядок завдань є істотним у складанні розкладів за допомогою евристики для випадку, коли завдання є нерівноцінними. Ефективність порядку завдань буде досліджена на прикладі щільного прогресуючого 1-машинного планування з перемиканнями без простою.

Методика реалізації. Для досягнення зазначеної мети проводиться обчислювальне дослідження з метою оцінки усередненого часу обчислення як для висхідного порядку, так і для спадного порядку дат запуску завдань. Приклади задачі планування завдань генеруються так, що розклади, які можна отримати тривіально, без евристики, не розглядаються.

Результати дослідження. У середньому спадний порядок завдань дає крихітну перевагу в часі обчислень. Ця перевага зменшується зі збільшенням кількості завдань. Декремент нагадує різкий експоненціальний спад. Фактична перевага настільки незначна, що навіть після розв'язування тривалих серій задач планування завдань заощаджений обчислювальний час не можна перерахувати в хвилинах, не кажучи вже про години, як це було у випадку рівноцінних завдань.

Висновки. Істотність порядку вводу завдань є набагато нижчою, ніж у випадку рівноцінних завдань. Ефективний порядок вводу завдань у евристиці теоретично існує, але його ефективність може бути практично використана лише при роботі з надзвичайно довгими рядами задач планування, де кількість завдань не повинна перевищувати 300.

Ключові слова: планування завдань на одній машині з перемиканнями; загальне 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 ПРИ ЭФФЕКТИВНОМ ПОРЯДКЕ ВВЕДЕНИЯ ЗАДАНИЙ В ЭВРИСТИКЕ

Проблематика. В постановке задачи минимизации общего 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.

Цель исследования. Цель состоит в установлении того, является ли порядок заданий значимым в составлении расписаний с помощью эвристики для случая, когда задания неравноценны. Эффективность порядка заданий будет исследована на примере плотного прогрессирующего 1-машинного планирования с переключениями без простоя.

Методика реализации. Для достижения указанной цели проводится вычислительное исследование с целью оценки усредненного времени вычисления как для восходящего порядка, так и для нисходящего порядка дат запуска заданий. Примеры задачи планирования заданий генерируются так, что расписания, которые можно получить тривиально, без эвристики, не рассматриваются.

Результаты исследования. В среднем нисходящий порядок заданий дает крошечное преимущество во времени вычислений. Это преимущество уменьшается с увеличением количества заданий. Декремент напоминает резкую экспоненциальную убыль. Фактическое преимущество настолько незначительно, что даже после решения длительных серий задач планирования заданий сэкономленное вычислительное время нельзя перечислить в минутах, не говоря уже о часах, как это было в случае равноценных заданий.

Выводы. Значимость порядка ввода заданий намного ниже, чем в случае равноценных заданий. Эффективный порядок ввода заданий в эвристике теоретически существует, но его эффективность может быть практически использована только при работе с чрезвычайно длинными рядами задач планирования, где количество заданий не должно превышать 300.

Ключевые слова: планирование заданий на одной машине с переключениями; общее 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 березня 2020 року

Прийнята до публікації
25 червня 2020 року