DOI: 10.20535/kpi-sn.2020.2.181869
UDC 519.161+519.687.1+004.02

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

HEURISTIC'S JOB ORDER EFFICIENCY IN TIGHT-TARDY PROGRESSIVE IDLING-FREE 1-MACHINE PREEMPTIVE SCHEDULING OF EQUAL-LENGTH JOBS

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 proved that an efficient job order can save significant computation time by using the Boolean linear programming model provided for finding schedules with the exactly minimal total tardiness.
Objective. The goal is to ascertain whether the job order input is significant in scheduling by using the heuristic. Job order efficiency will be studied on tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs.
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. Scheduling a few jobs is expectedly faster by ascending order, but this part is full of computational artifacts. 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.
Conclusions. In scheduling by using the heuristic, the job order input is indeed significant. The average relative difference does not exceed 1.5 % for 2 to 1000 jobs consisting up to 17 processing periods. For obtaining a statistically reliable computation speed advantage, it is better to consider no less than 250 jobs. As either the number of jobs or the number of job parts increases, the computation speed advantage may become unstable and eventually vanish. Nevertheless, the ascending job order can save a lot of computational time in the case of scheduling at least a few thousand jobs having just a few processing periods each. After solving thousands of such cases the saved time may be counted in hours.

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

Introduction

In scheduling, total tardiness is a measure of delays in executing certain operations. The problem of total tardiness minimization refers to attempts of optimizing time resources and minimizing costs, where jobs have no priorities. This problem has a great impact on dispatching processes like those at airports, water ports, railway stations, etc. [1, 2]. Besides, in computing, it is connected with optimizing parallelization of tasksets on multiprocessor systems [3, 4].

The exact minimization of total tardiness is possible just for a few jobs whose processing periods are not very long. The Boolean linear programming model and its variations are used for that [5, 6]. Nevertheless, the tasksets considered on multiprocessor systems may consist of a few hundreds or thousands of jobs. In this case, total tardiness is minimized by heuristics [7, 8]. The heuristic approach is the only means to deal with total tardiness minimization when the number of jobs and the numbers of their processing periods increase.

The heuristic based on the remaining available period and remaining processing period [8] is closely the best as of February 2020. It is an online scheduling algorithm which builds a sequence of jobs successively, without re-arranging them. Once a job is scheduled at a time moment, it will not be changed. So, the jobs already scheduled can be executed straightforwardly without waiting for the entire schedule. All the more so because the entire schedule can be an extremely long sequence of jobs [2, 4, 8, 9].

The accuracy of the heuristic is not ever perfect but, as the size of the job scheduling problem grows, the difference between the heuristic's total tardiness and the real minimum of total tardiness gradually vanishes [8]. A remarkable property of the heuristic is that it schedules just 2 jobs always at the 100 %

accuracy, not depending on in how many parts the job is divided (the "2/any" exception). Moreover, the heuristic is sufficiently accurate if no less than 7 jobs divided into no less than five parts each are scheduled (the "7/5" pattern). In general, as the job scheduling problem size grows, the inaccuracy of the heuristic drops [8].

Problem statement

The problem of total tardiness minimization considers job processing periods, release dates and due dates. All they are given as natural numbers. A partial case is when every job has the same length, i. e. the processing periods of all the jobs are equal. In setting a problem of minimizing total tardiness by the heuristic, there are two opposite ways to input the data. On one hand, the job release dates are given in ascending order. This is the common way of inputting the data into the algorithm. Strictly speaking, the release dates can be permuted as one likes or needs to satisfy some external conditions. Thus, on the other hand, the job release dates are given in descending order. Article [5] proves that scheduling a fewer jobs divided into a fewer job parts, using the Boolean linear programming model provided for finding schedules with the exactly minimal total tardiness, is executed on average faster by the descending job order. As the number of jobs increases along with increasing the number of their processing periods, the ascending job order becomes more efficient, although the computation time efficiency by both job orders is not so regular. Hence, the goal is to ascertain whether the job order input is significant in scheduling by using the heuristic. Job order efficiency will be studied on tight-tardy progressive idling-free single machine (1-machine) preemptive scheduling of equal-length jobs [5].

The heuristic based on remaining available and processing periods

Given N jobs, $N \in \mathbb{N} \setminus \{1\}$, having H processing periods each, their release dates

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

or

$$r_n = N - n + 1 \quad \forall\, n = \overline{1,\, N} \tag{2}$$

and due dates

$$d_n = r_n + H - 1 + b_n \quad \forall\, n = \overline{1,\, N} \tag{3}$$

for release dates (1) or

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

for release dates (2) by a random due date shift b_n, sum

$$\sum_{n=1}^{N} \max\{0,\, \theta(n;\, H) - d_n\} \tag{5}$$

is to be minimized, where

$$\theta(n;\, H) \in \{\overline{1,\, N \cdot H}\}$$

is a moment of completing job n. Random due date shifts are taken from vector [5]

$$\mathbf{B} = [b_n]_{1\times N} = \psi(H \cdot \Xi(1,\, N)) \tag{6}$$

so that

$$d_n \geq 1 \quad \forall\, n = \overline{1,\, N} \tag{7}$$

and at least one of inequalities

$$b_n - 1 \leq b_{n+1} \quad \forall\, n = \overline{1,\, N-1} \tag{8}$$

is violated, where operator $\Xi(1,\, N)$ returns a pseudorandom $1 \times N$ vector whose entries are drawn from the standard normal distribution (with zero mean and unit variance), and function $\psi(\xi)$ returns the integer part of number ξ (e. g., see [5, 9]). So, for a properly given due date shift vector (6), where condition (7) is true and condition (8) is false, due dates (3) set in the order corresponding to ascending order of the release dates (1) are

$$d_n = H + n - 1 + b_n \quad \forall\, n = \overline{1,\, N} \tag{9}$$

and due dates (4) set in the order corresponding to descending order of the release dates (2) are

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

Thus, components of due dates vector (3) are not given in non-descending order. This is done so because in the case of when

$$d_n \leq d_{n+1} \quad \forall\, n = \overline{1,\, N-1} \tag{11}$$

is true, a schedule ensuring the exactly minimal total tardiness is found trivially, without resorting to any algorithm or model (see Theorem 1 in [5]). Components of due dates vector (4) are not given in non-ascending order by symmetrical reasoning.

The heuristic based on remaining available and processing periods minimizes sum (5) by building stepwise a schedule $\tilde{\mathbf{S}} = [\tilde{s}_t]_{1\times(N\cdot H)}$ as time t progresses. Before the start,

$$q_n = H \quad \forall\, n = \overline{1,\, N}\,. \tag{12}$$

Then, for every set of available jobs

$$A(t) = \{i \in \{\overline{1,\, N}\} : r_i \le t \text{ and } q_i > 0\} \subset \{\overline{1,\, N}\} \tag{13}$$

the remaining available period is

$$b_i = \max\{0,\, d_i - t + 1\} \quad \forall\, i \in A(t) \tag{14}$$

and a subset

$$A^*(t) = \arg\max_{i\in A(t)} (\max\{q_i,\, b_i\})^{-1} \tag{15}$$

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

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

then

$$\tilde{s}_t = i^* \text{ by } q_{i^*}^{(\text{obs})} = q_{i^*} \text{ and } q_{i^*} = q_{i^*}^{(\text{obs})} - 1; \tag{16}$$

otherwise

$$A^*(t) = \{i_l^*\}_{l=1}^{L} \subset A(t) \subset \{\overline{1,\, N}\} \text{ by } L > 1, \tag{17}$$

whence

$$\tilde{s}_t = i_1^* \text{ by } q_{i_1^*}^{(\text{obs})} = q_{i_1^*} \text{ and } q_{i_1^*} = q_{i_1^*}^{(\text{obs})} - 1. \tag{18}$$

Assignment (18) executed by condition (17) for subset (15) implies that, in a case when there are two or more maximal decisive ratios in (15), the earliest job is preferred to be scheduled [8]. Thus, job n is completed after moment $\tilde{\theta}(n;\, H)$ if

$$\tilde{s}_{\tilde{\theta}(n;\, h)} = n \quad \forall\, h = \overline{1,\, H}$$

by

$$\tilde{\theta}(n;\, h) \in \{\overline{1,\, N\cdot H}\}$$

and

$$\tilde{\theta}(n;\, h) < \tilde{\theta}(n;\, h+1) \quad \text{for} \quad h = \overline{1,\, H-1}\,.$$

Finally, using formula (5), amount

$$\tilde{\vartheta}(N,\, H) = \sum_{n=1}^{N} \max\{0,\, \tilde{\theta}(n;\, H) - d_n\} \tag{19}$$

is an approximately minimal total tardiness that corresponds to schedule $\tilde{\mathbf{S}} = [\tilde{s}_t]_{1\times(N\cdot H)}$ returned by the heuristic.

Generation of the job scheduling problem instances

When due date shift vector (6) is properly generated by the definite numbers of jobs N and of job parts H, due dates (9) corresponding to ascending order and due dates (10) corresponding to descending order are calculated. Then an ascending order schedule by release dates (1) and due dates (9) is computed by the heuristic (12)—(19). Alternatively, a descending order schedule by release dates (2) and due dates (10) is computed as well.

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

$$\tau_{Asc}(N,\, H) = \frac{1}{C}\sum_{c=1}^{C} \tau_{Asc}(N,\, H,\, c) \tag{20}$$

and

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

In percentage terms, the relative difference between computation times (20) and (21) is

$$\eta(N,\, H) = 100 \cdot \frac{\tau_{Asc}(N,\, H) - \tau_{Desc}(N,\, H)}{\tau_{Asc}(N,\, H)}\,. \tag{22}$$

Relative difference (22) will be estimated over a natural rectangular lattice, which is formed by

$$N = \overline{2,\, 1000} \quad \text{and} \quad H = \overline{2,\, 17}\,. \tag{23}$$

It is assumed that the rectangular lattice formed by (23) is sufficient to obtain reliable statistics for unbiased estimation of relative difference (22) by setting number C at no less than 100. So, let $C = 100$ for scheduling more than 400 jobs. For scheduling between 251 and 400 jobs, the most appropriate number is $C = 200$. Further, let $C = 500$ for scheduling between 101 and 250 jobs. Finally, let $C = 1000$ for scheduling no more than 100 jobs. For quick reference, these numbers of the instances are visualized in Fig. 1.

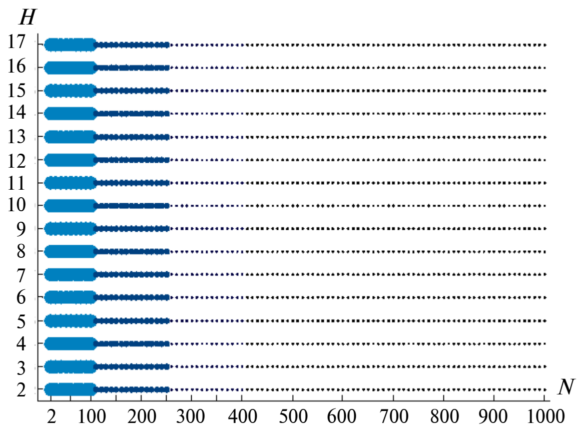

Fig. 1. The total numbers of the instances on the rectangular lattice formed by (23), where a greater number of instances is marked with a bigger circle of a lighter color

Computational study

The computational study is executed on CPU Intel Core i5-7200U@2.50 GHz using MATLAB R2018a. Fig. 2 shows all the 16 polylines (20) on the same plot versus the number of jobs to be scheduled. For every next H the plot of its polyline is higher. The computational artifacts (unexpected and non-predictable bad range fluctuations) are clearly seen. As the number of job processing periods increases, the averaged computation time by ascending order resembles more an exponential growth. The averaged computation time by descending order appears similarly (Fig. 3), although its computational artifacts are less striking.

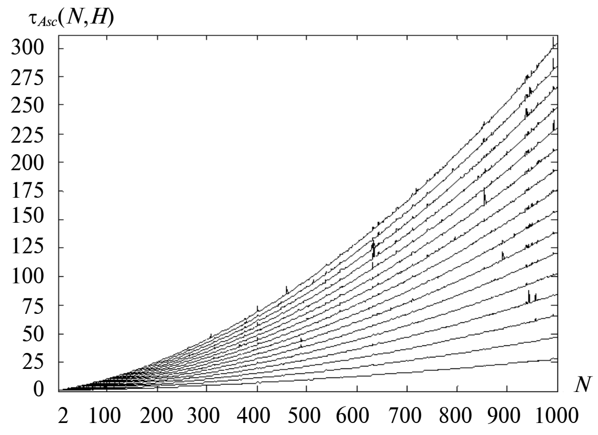

Fig. 2. The averaged computation time (ms) by ascending order for each $H = \overline{2, 17}$

Relative difference (22) between computation times (20) and (21) for each $H = \overline{2, 17}$ is presented in a fused plot in Fig. 4, where the number of processing periods increases from the left to the right downwards. The plot is an entire view of the 16 relative differences, in which all the fluctuations are preserved. There are 11 subplots which are clearly seen to be displaced downwards, i. e. their maximal positive fluctuations are greater than the respective maximal negative fluctuations. However, this does not mean that computing by descending order is faster on average here. Those three subplots clearly seen to be displaced upwards do not imply also an advantage of computing by ascending order. The two subplots in the second row, which seem to have no displacement, cannot be used for implications as well.

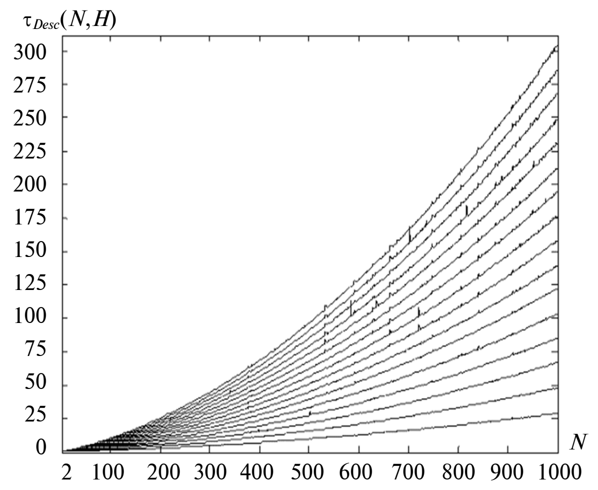

Fig. 3. The averaged computation time (ms) by descending order for each $H = \overline{2, 17}$

A refined version of the fused plot in Fig. 4 is presented in Fig. 5 using the same presentation style, by only cutting values of every polyline to the range of −5 % to 5 %. The horizontal zero level line is put on every subplot: if, within some interval of N , the range fluctuations are below the zero level, then it means that the ascending job order is faster over this interval; otherwise, if they are above the zero level, computing by descending order is faster. Visually, only polylines on the first row (i. e., for $H = 2$ and $H = 3$) are seen to be certainly lower (except for a few places which could be considered as microartifacts) than the zero level for $N > 100$. As the number of job processing periods increases, it is hard to ascertain the computation speed advantage. Nevertheless, it is well seen in every subplot in Fig. 5 that scheduling up to 90 jobs (except for scheduling a few jobs) is faster by descending order. Then, up to approximately 250 jobs, the ascending job order results in shorter computation times. Since the point of about 250 jobs, the advantage trend (of either ascending or descending order) appears more stable.

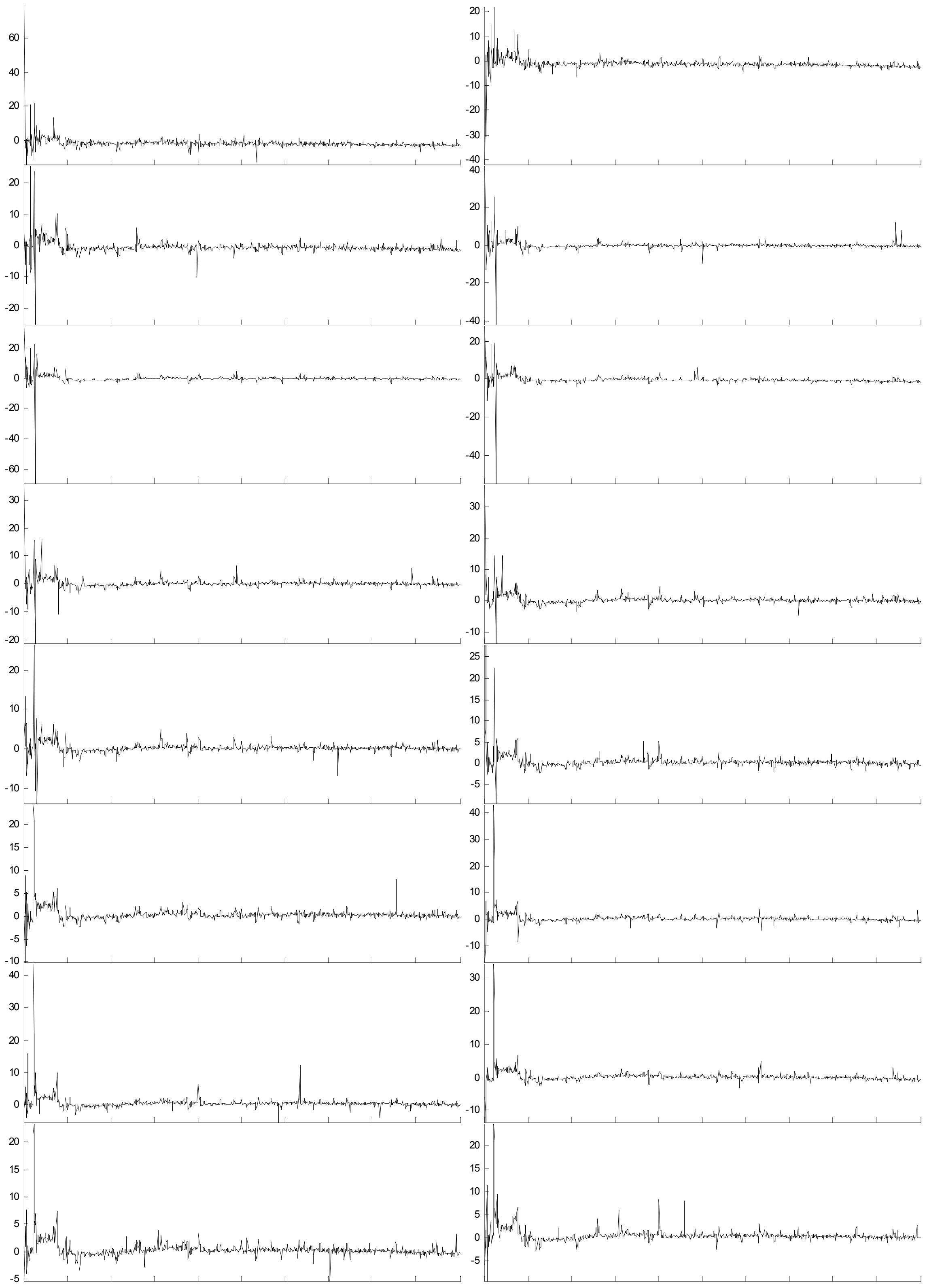

Fig. 4. A fused plot of relative difference (22) between computation times (20) and (21) for each $H = \overline{2, 17}$ (which increases from the left to the right downwards) versus $N = \overline{2, 1000}$

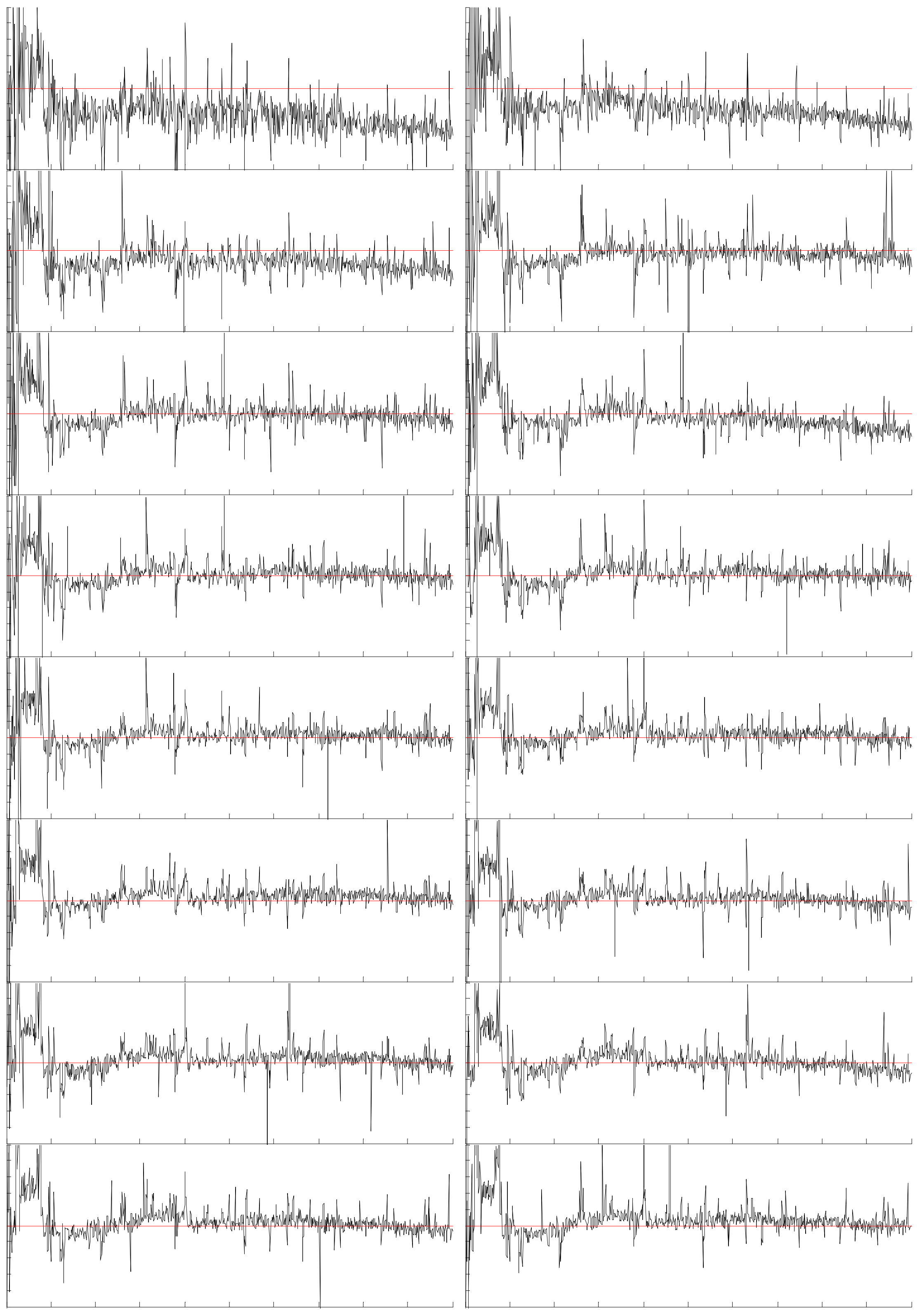
Fig. 5. Relative difference (22) for each $H = \overline{2, 17}$ (which increases from the left to the right downwards) versus $N = \overline{2, 1000}$, where every polyline is cut to the range of $-5\,\%$ to $5\,\%$

To see clearer what advantage or disadvantage there is (with respect to the zero level line), consider relative differences (22), averaged over the number of jobs. This is a set of 16 values

$$\bar{\eta}(H) = \frac{1}{999} \cdot \sum_{N=2}^{1000} \eta(N, H) \quad \text{for} \quad H = \overline{2, 17} \quad (24)$$

shown in the barred plot in Fig. 6. Now, it is certainly seen that scheduling jobs consisting of two to four parts is on average faster by ascending order. Scheduling jobs consisting of more than seven parts is on average faster by descending order. The range of jobs consisting of five to seven parts is an interval of uncertainty, where the advantage of the ascending job order is not so reliable. Nevertheless, this is a place wherein the advantage of the ascending job order vanishes and the advantage of the descending job order appears, being weaker though.

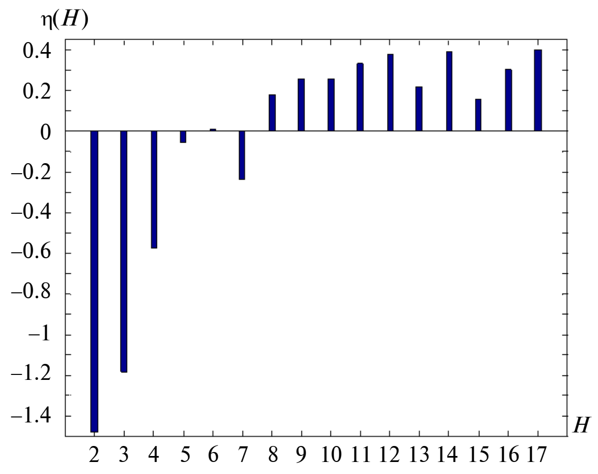

Fig. 6. The average of every polyline of relative difference (22) for each $H = \overline{2, 17}$

With the heuristic, neither the advantage of the ascending job order, nor that of the descending job order is significantly strong. For example, scheduling 1000 jobs divided into 17 processing periods each (Fig. 7 shows the due dates for this example) by ascending order takes 327.7 ms, whereas the same task is computed by descending order in 312.2 ms. In fact, the difference is not very great (just 15.5 ms) but if the similar task is computed for, say, 10000 times, then the descending job order allows saving up to 155 seconds. Another example, with 1000 jobs divided into 50 processing periods each (Fig. 8 shows the due dates for this example), is not so promising: the ascending and descending job orders take about 931.8 ms and 929.3 ms, respectively. So, here the difference is just about 2.5 ms and thus the descending job order allows saving slightly more than 25 seconds after 10000 rounds of such a task.

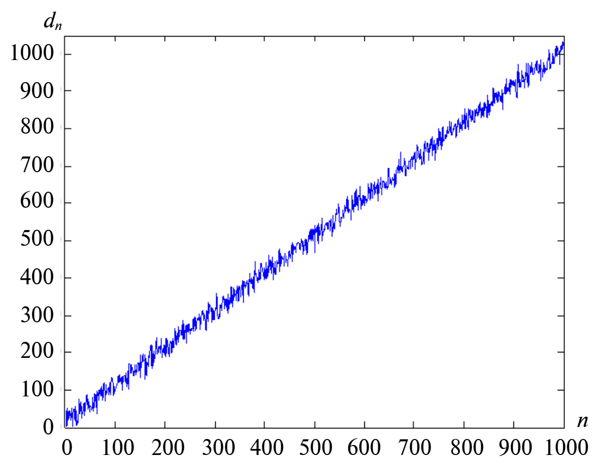

Fig. 7. The due dates corresponding to the release dates in ascending order for an example of scheduling 1000 jobs divided into 17 processing periods each; despite the due dates are not given in non-descending order, they roughly are linearly-increasing on average

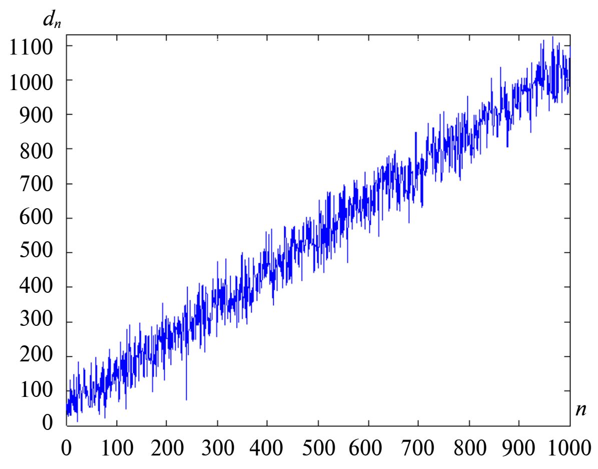

Fig. 8. The due dates corresponding to the release dates in ascending order for an example of scheduling 1000 jobs divided into 50 processing periods each; the due dates roughly are linearly-increasing on average but they appear more scattered (compared to those in Fig. 7)

Other scheduling examples similar to the mentioned do not necessarily lead to that the descending job order allows saving some time, whichever it is. As the number of job processing periods increases, there is no certain proof that the average of relative difference (22) shown in Fig. 6 will be still positive. As the number of jobs increases (with the same great number of job processing periods), the descending job order is likely to lose its advantage.

Discussion

Unfortunately, Fig. 6 does not show an entire and reliable panorama of which job order has a computation speed advantage. Moreover, these bars are just a result which should be treated as "on average" only. Thus, scheduling jobs having the minimal number of processing periods is averagely almost 1.5 % faster by ascending order, whereas scheduling 30 to 70 jobs is 1.5 % to 2.5 % faster by descending order (see this in the subplotted polylines of Fig. 5). Therefore, the bars in Fig. 6 will really work for multiple solvings (at least, a few tens of the heuristic's applications are required to establish a stable statistical ratio).

Another nuance is that the computation speed advantage seems to vanish as either the number of jobs or the number of job processing periods increases. Thus, scheduling 1000 jobs divided into 30 processing periods each (the due dates for this example in Fig. 9 appear less scattered than those in Fig. 8, and more scattered than those in Fig. 7) by the ascending and descending job orders takes about 557.5 ms and 557.1 ms, respectively. Such a tiny difference can be really counted as a statistically negligible. After increasing the number of processing periods to 100 each (the due dates for this example in Fig. 10 appear already badly scattered), the job order advantage changes: the ascending and descending job orders take about 1846.8 ms and 1864.1 ms, respectively; so now the ascending job order saves almost 173 seconds after 10000 rounds of such a task.

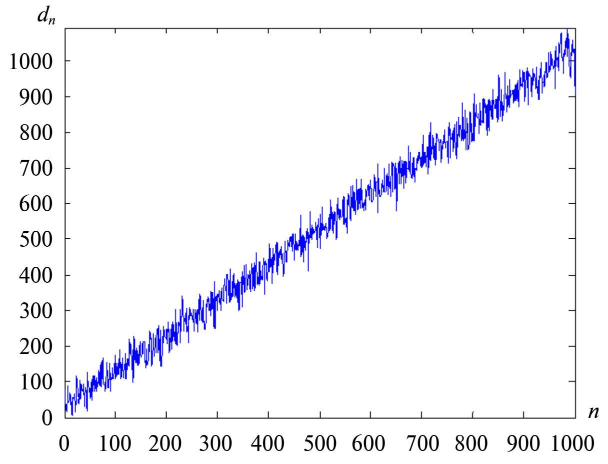

Fig. 9. The on-average-linearly-increasing due dates corresponding to the release dates in ascending order for an example of scheduling 1000 jobs divided into 30 processing periods each

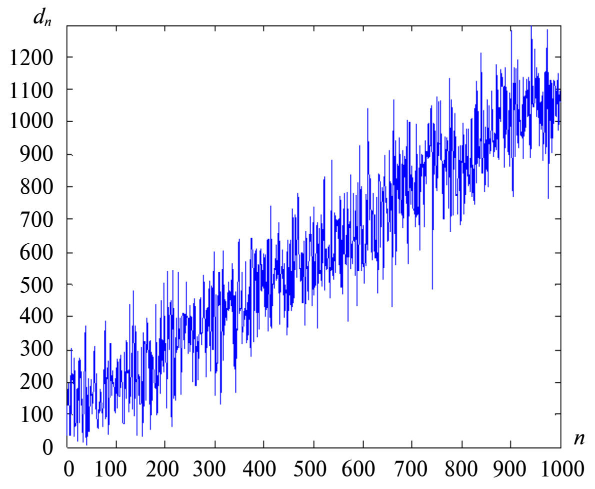

Fig. 10. The on-average-linearly-increasing due dates corresponding to the release dates in ascending order for an example of scheduling 1000 jobs divided into 100 processing periods each; in this example, unlike the examples with 17, 50, 30 processing periods, the ascending job order saves almost 173 seconds after 10000 rounds of such a task)

Nevertheless, the advantage in computing schedules of huger amounts of jobs having a few processing periods each can be pretty impressive. Thus, a task of scheduling 1000 jobs with two processing periods is solved by the ascending and descending job orders taken about 31.7 ms and 32.4 ms, respectively (the relative difference is 2.32 %, which does not much contradict the longest bar in Fig. 6). For 10000 jobs, these computation times become about 1934.1 ms and 2325.9 ms, respectively, and here the relative difference is a way huger: it is 20.26 %, so, after 10000 rounds of such a task, the ascending job order saves more than an hour (!). Furthermore, for 10000 jobs with three processing periods (it is better to pass to seconds now), these computation times become about 3.57931 seconds and 4.0849 seconds, respectively, and the ascending job order saves more than 1.4044 hours (!).

Conclusions

By using the heuristic based on remaining available and processing periods, the job order input is indeed significant in tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs. The relative difference between computation times by the job release dates given in ascending order and given in descending order can achieve up to 5 % and even more, although then it is considered as a computational artifact whose probability is small. The average relative difference does not exceed 1.5 % for 2 to 1000 jobs consisting up to 17 processing periods.

Regardless of how many processing periods the job has, scheduling a few jobs is expectedly faster by ascending order, but this part is full of computational artifacts. Then, scheduling up to 90 jobs is expectedly faster by descending order. For the number of jobs between roughly 90 and 250, the ascending job order again results in shorter computation times.

On average, where it is better to consider no less than 250 jobs, the computation speed advantage of the ascending job order gradually disappears as the number of job processing periods increases and then the descending job order becomes faster. The break is at about five to seven parts of the job. However, as either the number of jobs or the number of job parts increases, the computation speed advantage may become unstable and eventually vanish. Nevertheless, the ascending job order can save a lot of computational time in the case of scheduling at least a few thousand jobs having just a few processing periods each. After solving thousands of such cases the saved time may be counted in hours. Whether the obtained properties keep in the similar manner for the case of jobs having different number of processing periods is a matter of a further research.

References

[1] J.C. Price and J.S. Forrest, *Practical Airport Operations, Safety, and Emergency Management.* Butterworth-Heinemann, 2016, 630 p. doi: 10.1016/C2013-0-15991-0

[2] 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

[3] H. Alhussian *et al.*, "An unfair semi-greedy real-time multiprocessor scheduling algorithm", *Computers & Electrical Engineering*, vol. 50, pp. 143–165, 2016. doi: 10.1016/j.compeleceng.2015.07.003

[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, "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.2020.1.164804

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

[7] 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

[8] V.V. Romanuke, "Accurate total weighted tardiness minimization in tight-tardy progressive single machine scheduling with preemptions by no idle periods", *KPI Sci. News*, no. 5-6, pp. 26–42, 2019. doi: 10.20535/kpi-sn.2019.5-6.178016

[9] V.V. Romanuke, "Accuracy of a heuristic for total weighted completion time minimization in preemptive single machine scheduling problem by no idle time intervals", *KPI Sci. News*, no. 3, pp. 52–62, 2019. doi: 10.20535/kpi-sn.2019.3.164804

В.В. Романюк

ЕФЕКТИВНІСТЬ ПОРЯДКУ ЗАВДАНЬ У ЕВРИСТИЦІ ЩІЛЬНОГО ПРОГРЕСУЮЧОГО 1-МАШИННОГО ПЛАНУВАННЯ РІВНОЦІННИХ ЗАВДАНЬ ІЗ ПЕРЕМИКАННЯМИ БЕЗ ПРОСТОЮ

Проблематика. У постановці задачі мінімізації загального запізнювання за евристикою на основі використання залишкового наявного ресурсу та залишкового періоду до обробки існують два протилежних способи введення даних: дати запуску завдань задаються в порядку зростання або спадання. Нещодавно було доведено, що ефективний порядок завдань може заощадити значний час обчислень при використанні моделі булевого лінійного програмування для пошуку розкладів зі строго мінімальним загальним запізнюванням.

Мета дослідження. Метою є встановлення того, чи порядок завдань є суттєвим у складанні розкладів за допомогою евристики. Ефективність порядку завдань буде досліджена на прикладі щільного прогресуючого 1-машинного планування рівноцінних завдань із 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 випадку кількох завдань їх планування виконується очікувано швидше за висхідного порядку, але у цій частини забагато обчислювальних артефактів. Планування від 30 до 70 завдань на 1,5–2,5 % швидше за спадного порядку. Однак планування до 90 завдань, як очікується, все одно буде швидше за спадним порядком, хоча існує ризик втрати цієї переваги. Для кількості завдань приблизно від 90 до 250 висхідний порядок завдань знову приводить до скорочення часу обчислень. Починаючи з точки близько 250 завдань, тенденція переваги (висхідного або спадного порядку) виглядає більш стабільною.

Висновки. При плануванні за допомогою евристики введення порядку завдань є дійсно суттєвим. Середня відносна різниця не перевищує 1,5 % для випадку від 2 до 1000 завдань, що мають до 17 періодів до обробки. Для отримання статистично достовірної переваги швидкості обчислення краще розглядати випадки з не менш як 250 завданнями. Зі збільшенням кількості завдань або кількості частин одного завдання перевага швидкості обчислення може стати нестабільною і з часом зникнути. Водночас висхідний порядок завдань може заощадити багато обчислювального часу у випадку складання розкладів принаймні декількох тисяч завдань, що мають лише кілька періодів до обробки. Після розв'язання тисяч таких випадків заощаджений час може нараховувати години.

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

В.В. Романюк

ЭФФЕКТИВНОСТЬ ПОРЯДКА ЗАДАНИЙ В ЭВРИСТИКЕ ПЛОТНОГО ПРОГРЕССИРУЮЩЕГО 1-МАШИННОГО ПЛАНИРОВАНИЯ РАВНОЦЕННЫХ ЗАДАНИЙ С ПЕРЕКЛЮЧЕНИЯМИ БЕЗ ПРОСТОЯ

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 доказано, что эффективный порядок заданий может сэкономить значительное время вычислений при использовании модели булевого линейного программирования для поиска расписаний со строго минимальным общим запаздыванием.

Цель исследования. Цель состоит в установлении того, является ли порядок заданий существенным в составлении расписаний с помощью эвристики. Эффективность порядка заданий будет исследована на примере плотного прогрессирующего 1-машинного планирования равноценных заданий 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 случае нескольких заданий их планирование выполняется ожидаемо быстрее при восходящем порядке, но в этой части много вычислительных артефактов. Планирование от 30 до 70 заданий на 1,5–2,5 % быстрее при нисходящем порядке. Однако планирование до 90 заданий, как ожидается, все равно будет быстрее при нисходящем порядке, хотя существует риск потери этого преимущества. Для количества заданий примерно от 90 до 250 восходящий порядок заданий снова приводит к сокращению времени вычислений. Начиная с точки около 250 заданий, тенденция преимущества (восходящего или нисходящего порядка) выглядит более стабильной.

Выводы. При планировании с помощью эвристики введение порядка заданий действительно является существенным. Средняя относительная разность не превышает 1,5 % для случая от 2 до 1000 заданий, имеющих до 17 периодов к обработке. Для получения статистически достоверного преимущества скорости вычисления лучше рассматривать случаи с не менее чем 250 заданиями. С увеличением количества заданий или количества частей одного задания преимущество скорости вычисления может стать нестабильным и со временем исчезнуть. Тем не менее восходящий порядок заданий может сэкономить много вычислительного времени в случае составления расписаний по крайней мере нескольких тысяч заданий, имеющих лишь несколько периодов к обработке. После решения тысяч таких случаев сэкономленное время может насчитывать часы.

Ключевые слова: планирование заданий на одной машине с переключениями; равноценные задания; 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
28 жовтня 2019 року

Прийнята до публікації
05 червня 2020 року