UDC 519.161+519.852+519.687.1

TOTAL WEIGHTED TARDINESS EXACT MINIMIZATION BY EFFICIENTLY INPUTTING JOBS TO TIGHT-TARDY PROGRESSIVE SINGLE MACHINE SCHEDULING WITH IDLING-FREE PREEMPTIONS

Romanuke V. V.

O. S. Popov Odesa National Academy of Telecommunications,
1 Kuznechna St., Odesa, 65029, Ukraine.
romanukevadimv@gmail.com

ТОЧНА МІНІМІЗАЦІЯ ЗАГАЛЬНОГО ЗВАЖЕНОГО ЗАПІЗНЮВАННЯ ЗА ЕФЕКТИВНОГО ВВОДУ ЗАВДАНЬ У ЩІЛЬНЕ ПРОГРЕСУЮЧЕ ОДНОМАШИННЕ ПЛАНУВАННЯ З ПЕРЕМИКАННЯМИ БЕЗ ПРОСТОЮ

Романюк В. В.

Одеська національна академія зв'язку ім. О. С. Попова,
65029, Україна, м. Одеса, вул. Кузнечна, 1.
romanukevadimv@gmail.com

ТОЧНАЯ МИНИМИЗАЦИЯ ОБЩЕГО ВЗВЕШЕННОГО ЗАПАЗДЫВАНИЯ ПРИ ЭФФЕКТИВНОМ ВВЕДЕНИИ ЗАДАНИЙ В ПЛОТНОЕ ПРОГРЕССИРУЮЩЕЕ ОДНОМАШИННОЕ ПЛАНИРОВАНИЕ С ПЕРЕКЛЮЧЕНИЯМИ БЕЗ ПРОСТОЯ

Романюк В. В.

Одесская национальная академия связи им. А. С. Попова,
65029, Украина, г. Одесса, ул. Кузнечная, 1.
romanukevadimv@gmail.com

Abstract. *A schedule ensuring the exactly minimal total weighted tardiness can be found with the respective integer linear programming problem. An open question is whether the exact schedule computation time changes if the job release dates are input to the model in reverse order. The goal is to ascertain whether the job order in tight-tardy progressive single machine scheduling with idling-free preemptions influences the speed of computing the exact solution. The jobs can have both different lengths and different priority weights. The Boolean linear programming model provided for finding schedules with the minimal total weighted tardiness is used. 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. Then the relative difference between the computation times is to be estimated and treated. The job order influences the speed of computing the exact solution but it is hardly possible to predict it. The matter is that the jobs can have both different lengths and different priority weights, that additionally "dithers" the respective job order input leaving only release dates which are always given in that order. It has been revealed that scheduling 3 or 4 jobs is executed on average faster by the descending job order input. However, the expected acceleration by the descending job order input cannot be estimated with a great confidence factor. This result disproves a possibility to manipulate the job order for obtaining schedules more efficiently in a single job scheduling problem or in a few such problems. Only if the job scheduling problems by 3 or 4 jobs resemble the descending job order input, and there are thousands of such problems, then it is recommended to use that job order input. Inputting jobs efficiently by the descending job order style is still possible when different job lengths and different priority weights are both scattered not much, that would be closer to the case of total tardiness by equal job lengths and equal priority weights. This case is the most promising one, where the descending job order computation time can be shorter up to 10 % and more in scheduling 2 to 6 jobs divided into two to four or even five parts each.*

Keywords: *job scheduling; preemptive single machine scheduling; exact model; total weighted tardiness; computation time; ascending job order; descending job order.*

Анотація. *Розклад, що забезпечує строго мінімальне загальне зважене запізнювання, можна знайти за відповідною цілочисловою задачею лінійного програмування. Відкритим є питання про те, чи змінюється час обчислення точного розкладу, якщо дати запуску завдань вводяться у модель у зворотному порядку. Мета полягає у тому, щоб встановити, чи впливає на швидкість обчислення точного розв'язку порядок завдань у щільному прогресуючому одномашинному плануванні з перемиканнями без простою. Завдання можуть мати різні об'єми та різні ваги пріоритетів. Для пошуку розкладів з мінімальним загальним зваженим запізнюванням*

використовується модель булевого лінійного програмування. 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. Далі відносна різниця між часами обчислень має бути оцінена й опрацьована. Порядок завдань впливає на швидкість обчислення точного розв'язку, але це навряд чи є прогнозованим. Причиною цього є те, що завдання можуть мати різні об'єми та різні ваги пріоритетів, що додатково "розмиває" відповідний порядок вводу завдань, залишаючи лише дати запуску завдань, котрі завжди подаються у цьому порядку. Виявлено, що планування розкладу 3-х або 4-х робіт виконується у середньому швидше за спадного порядку вводу завдань. Однак очікуване прискорення за спадним порядком вводу завдань не може бути оціненим з високою достовірністю. Цей результат спростовує можливість маніпулювати порядком завдань з метою більш ефективного отримання розкладів в одиночному випадку задачі планування розкладу або у декількох таких випадках. Тільки якщо задачі планування розкладів з 3-ма або 4-ма завданнями подібні до спадного порядку вводу завдань і подаються тисячі таких задач, то лише тоді використання такого порядку вводу завдань рекомендовано. Вводити завдання ефективно у такому стилі все ще можливо тоді, коли розкид різних об'ємів завдань та різних ваг пріоритетів не є сильним, що стає ближчим до випадку загального запізнювання за рівних об'ємів завдань і рівних ваг пріоритетів. Цей випадок є найбільш привабливим, де час обчислень за спадного порядку завдань може скорочуватися до 10 % і більше при плануванні розкладів від 2-х до 6-ти завдань, що мають у складі від двох до чотирьох або навіть п'яти частин кожне.

Ключові слова: *планування завдань; планування на одній машині з перемиканнями; точна модель; загальне зважене запізнювання; час обчислення; висхідний порядок завдань; спадний порядок завдань.*

Аннотация. *Расписание, обеспечивающее строго минимальное взвешенное общее запаздывание, можно найти по соответствующей целочисленной задаче линейного программирования. Открытым является вопрос о том, меняется ли время вычисления точного расписания, если даты запуска заданий вводятся в модель в обратном порядке. Цель состоит в том, чтобы установить, влияет ли на скорость вычисления точного решения порядок заданий в плотном прогрессирующем одномашинном планировании с переключениями без простоя. Задания могут иметь различные объёмы и различные веса приоритетов. Для поиска расписаний с минимальным общим взвешенным запаздыванием используется модель булевого линейного программирования. Для достижения указанной цели проводится вычислительное исследование с целью оценки усреднённого времени вычисления как для восходящего порядка, так и для нисходящего порядка дат запуска заданий. Далее относительная разность между временами вычислений должна быть оценена и обработана. Порядок заданий влияет на скорость вычисления точного решения, но это едва ли предсказуемо. Причина состоит в том, что задания могут иметь различные объёмы и различные веса приоритетов, что дополнительно "размывает" соответствующий порядок введения заданий, оставляя лишь даты запуска заданий, которые всегда подаются в указываемом порядке. Обнаружено, что планирование расписания 3-х или 4-х заданий выполняется в среднем быстрее при нисходящем порядке введения заданий. Однако ожидаемое ускорение при нисходящем порядке введения заданий не может быть оценено с высокой достоверностью. Этот результат опровергает возможность манипулировать порядком заданий с целью более эффективного получения расписаний в единичном случае задачи планирования расписания или в нескольких таких случаях. Только если задачи планирования расписаний с 3-мя или 4-мя заданиями подобны нисходящему порядку введения заданий и подаются тысячи таких задач, то лишь тогда использование такого порядка введения заданий рекомендовано. Вводить задания эффективно в таком стиле всё ещё возможно тогда, когда разброс различных объёмов заданий и различных весов приоритетов не является ощутимым, что становится ближе к случаю общего запаздывания при равных объёмах заданий и равных весах приоритетов. Этот случай является наиболее привлекательным, где время вычислений при нисходящем порядке заданий может сокращаться до 10 % и более при планировании расписаний от 2-х до 6-ти заданий, которые имеют в составе от двух до четырёх или даже пяти частей каждое.*

Ключевые слова: *планирование заданий; планирование на одной машине с переключениями; точная модель; общее взвешенное запаздывание; время вычисления; восходящий порядок заданий; нисходящий порядок заданий.*

Job order efficiency in the idling-free preemptive scheduling

A schedule ensuring the exactly minimal total weighted tardiness can be found with the respective integer linear programming problem involving the branch-and-bound approach [1, 2]. A class of tight-tardy progressive single machine scheduling with idling-free preemptions exists, in which release dates are set at non-repeating integers from 1 through the total number of jobs, and due dates are tightly set after the respective release dates (although sometimes a few jobs can be completed without tardiness). An open question is whether the exact schedule computation time changes if the release dates are input to the model in reverse order [3, 4]. The ascending job order implies inputting the release dates in ascending (i. e., starting from 1) order, and the descending job

order implies inputting the release dates in descending (i. e., starting from the last job) order.

The first attempt was made in article [5] which showed that a possibility exists to find schedules more efficiently by equal-length jobs. For instance, schedules of 5 jobs consisting of two processing periods each were found on average by 14.67 % faster for the descending job order. In another example of 7 three-parted jobs, an optimal schedule was found on average in 69.51 seconds by the ascending job order, whereas the descending job order took just 36.52 seconds to find it, saving thus 32.99 seconds. In general, article [5] revealed that, in the case of equal-length jobs, scheduling a fewer jobs divided into a fewer job parts 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. However, the computation time efficiency by both job orders tends to be irregular.

In the case of different job lengths studied in article [6], scheduling 2 to 5 jobs is executed on average faster by the descending job order input, where 1 % to 3 % speed-up is expected. Further increment of the number of jobs to be scheduled cannot guarantee any speed-up even on average. This result is similar to that in the case of equal-length jobs, and there is no regularity in such an efficient job order input. Therefore, article [5] revealed that, without any assurance for a single job scheduling problem, the efficient exact minimization of total tardiness by the descending job order input must be treated as on average only.

So, now the question is whether similar conclusions and properties are still peculiar to the general case when the jobs have different lengths and different priority weights [7, 8]. Will the efficiency of total weighted tardiness exact minimization by a job order input continue degrade (caused by applying priority weights)? How much time can be saved while computing exact schedules? These questions will be answered by a study similar to the studies in [5] and [6], but final recommendations aggregating all the studies on the class of tight-tardy progressive single machine scheduling with idling-free preemptions are to be formulated as well.

The goal and tasks to achieve it

The goal is to ascertain whether the job order in tight-tardy progressive single machine scheduling with idling-free preemptions influences the speed of computing the exact solution. The jobs can have both different lengths and different priority weights. Just as it was in [5] and [6], here the Boolean linear programming model provided for finding schedules with the minimal total weighted tardiness will be used. To achieve the said goal, a computational study should be carried out with a purpose to estimate the averaged computation time for both ascending and descending orders. For this, a pattern of generating instances of the job scheduling problem will be suggested similarly to that how it was done in articles [5] and [6]. Then the relative difference between the computation times is to be estimated and treated. The research result is expected to either reveal or disprove a possibility to manipulate the job order for obtaining schedules more efficiently, which will be the finalization of the research results in articles [5] and [6]. Eventually, the research results of all the three studies are to be combined and summarized.

Minimal total weighted tardiness by the varying number of processing periods

Every job is associated with its number of processing periods, priority weight, release date, and due date. Let job n be of H_n processing periods, and w_n is its priority weight, r_n is its release date, d_n is its due date, where $n = \overline{1, N}$ by the total number of jobs N, $N \in \mathbb{N} \setminus \{1\}$. Integer r_n is the time moment, at which job n becomes available for processing [7, 9]. Weights $\{w_n\}_{n=1}^{N}$ are positive integers. All the time moments and the processing periods are measured in the same time units, and thus they are synchronized.

Vector of processing periods (or job lengths)

$$\mathbf{H} = \left[H_n\right]_{1\times N} \in \mathbb{N}^N \tag{1}$$

associated with job priority weights

$$\mathbf{W} = \left[w_n \right]_{1 \times N} \in \mathbb{N}^N \tag{2}$$

does not have any specific constraints applied to it. Unlike vector (1), vector of release dates

$$\mathbf{R} = \left[r_n \right]_{1 \times N} \in \mathbb{N}^N \tag{3}$$

is constrained depending on the job order input and the requirement of that job preemptions be idling-free. If the release dates are given in ascending order then

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

If the release dates are given in descending order then

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

Vector of due dates

$$\mathbf{D} = \left[d_n \right]_{1 \times N} \in \mathbb{N}^N \tag{6}$$

is not purely random as the due dates are tightly set after the release dates, in whichever order they are given:

$$d_n = r_n + H_n - 1 + b_n \quad \forall n = \overline{1,\ N} \tag{7}$$

for ascending order and

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

for descending order, where b_n is a random due date shift

$$b_n = \psi\left(H_n \cdot \zeta \right) \ \text{ for } \ n = \overline{1,\ N} \tag{9}$$

with a pseudorandom number ζ drawn from the standard normal distribution (with zero mean and unit variance), and function $\psi\left(\xi\right)$ returning the integer part of number ξ (e. g., see [1, 3, 5, 6]). Due date shifts (9) are generated until

$$d_n \geqslant 1 \quad \forall n = \overline{1,\ N}\,. \tag{10}$$

If

$$H_n \leqslant H_{n+1} \ \text{ and } \ d_n \leqslant d_{n+1} \ \text{ and } \ w_n \geqslant w_{n+1} \quad \forall n = \overline{1,\ N-1} \tag{11}$$

for the ascending job order input, then due date shifts (9) are re-generated as well. Symmetrically, if

$$H_n \geqslant H_{n+1} \text{ and } d_n \geqslant d_{n+1} \text{ and } w_n \leqslant w_{n+1} \quad \forall n = \overline{1,\ N-1} \tag{12}$$

for the descending job order input, then due date shifts (9) are generated once again until such shared monotonicity is broken [6]. This is done so because in the case of when either inequalities (11) or inequalities (12) are true, a schedule ensuring the exactly minimal total weighted tardiness is found trivially, without resorting to any algorithm or model: if (11) are true for the ascending job order input, an optimal schedule is composed by arranging jobs from the earliest one to the latest one; if (12) are true for the descending job order input, an optimal schedule is composed by arranging jobs from the latest one to the earliest one [6]. This was proved for the case of equal-length jobs in article [5]. A more general case, when jobs can have different processing periods, was proved in article [6]. Obviously, associating jobs with their priority weights by (11) or (12) cannot violate the schedule triviality: a more important job (whose weight is greater) can always be released earlier, and thus its weighted tardiness is minimized.

Once due date shifts (9) are given properly, due dates (7) set in the order corresponding to ascending order of the release dates (4) are

$$d_n = H_n + n - 1 + b_n \quad \forall n = \overline{1,\ N} \tag{13}$$

and due dates (8) set in the order corresponding to descending order of the release dates (5) are

$$d_n = N + H_n - n + b_{N-n+1} \quad \forall n = \overline{1,\ N}\,. \tag{14}$$

The length of the schedule is

$$T = \sum_{n=1}^{N} H_n\,. \tag{15}$$

The goal is to minimize the total weighted tardiness through schedule's length (15), i. e. to schedule N jobs so that sum

$$\sum_{n=1}^{N} w_n \cdot \max\left\{0,\ \theta\left(n;\ H_n\right) - d_n\right\} \tag{16}$$

would be minimal, where job n is completed after moment $\theta\left(n;\ H_n\right)$, which is

$$\theta\left(n;\ H_n\right) \in \left\{\overline{1,\ T}\right\}.$$

This goal is equivalent to finding such decision variables which minimize sum [6, 10]

$$\sum_{n=1}^{N} \sum_{h_n=1}^{H_n} \sum_{t=1}^{T} \lambda_{nh_nt} x_{nh_nt}\,, \tag{17}$$

where x_{nht} is the decision variable about assigning the h-th part of job n to time moment t: $x_{nht} = 1$ if it is assigned; $x_{nht} = 0$ otherwise. The triple-indexed weights (these ones are not the job

priority weights)

$$\left\{\left\{\left\{\lambda_{nh_nt}\right\}_{t=1}^{T}\right\}_{h_n=1}^{H_n}\right\}_{n=1}^{N}$$

are calculated as follows:

$$\lambda_{nh_nt} = 0 \tag{18}$$

by

$$r_n - 1 + h_n \leqslant t \leqslant T - H_n + h_n \quad \forall h_n = \overline{1,\, H_n - 1} \tag{19}$$

and

$$\lambda_{nh_nt} = \alpha \tag{20}$$

by a sufficiently great positive integer α (similar to the meaning of infinity, i. e. it is an infinity "substitute" for real-practice calculations) when (19) is not true;

$$\lambda_{nH_nt} = 0 \tag{21}$$

by

$$r_n - 1 + H_n \leqslant t \leqslant d_n \tag{22}$$

and

$$\lambda_{nH_nt} = (t - d_n) w_n \tag{23}$$

by

$$d_n < t \leqslant T \tag{24}$$

and

$$\lambda_{nH_nt} = \alpha \tag{25}$$

when both (22) and (24) are not true. In (20) and (25), for instance,

$$\alpha = \sum_{n=1}^{N} \sum_{t=1}^{T} w_n t \tag{26}$$

can be used [1, 5, 6]. The decision variables constraints are as follows:

$$x_{nh_nt} \in \{0,1\} \text{ by } n=\overline{1,N} \text{ and } h_n=\overline{1,H_n} \text{ and } t=\overline{1,T}, \tag{27}$$

$$\sum_{t=1}^{T} x_{nh_nt}=1 \text{ by } n=\overline{1,N} \text{ and } h=\overline{1,H_n}, \tag{28}$$

$$\sum_{n=1}^{N}\sum_{h=1}^{H_n} x_{nh_nt}=1 \text{ by } t=\overline{1,T}, \tag{29}$$

$$\sum_{j=t+1}^{T}\sum_{h_n=1}^{H_n-1} x_{nh_nj}+H_n x_{nH_nt} \leqslant H_n \text{ by } n=\overline{1,N} \text{ and } t=\overline{1,T-1}. \tag{30}$$

An optimal job schedule

$$\mathbf{S}^*=\left[s_t^*\right]_{1\times T} \text{ by } s_t^* \in \left\{\overline{1,N}\right\} \text{ for every } t=\overline{1,T} \tag{31}$$

is found by a set of the decision variables at which sum (17) is minimal, where

$$s^*_{\theta^*(n;h)}=n \quad \forall h_n=\overline{1,H_n} \text{ by } \theta^*(n;h_n)\in\left\{\overline{1,T}\right\}$$
$$\text{and } \theta^*(n;h_n)<\theta^*(n;h_n+1) \text{ for } h_n=\overline{1,H_n-1}.$$

Thus, $\theta^*(n;H_n)$ is a moment after which job n is completed, and, according to sum (16),

$$\vartheta^*(N)=\sum_{n=1}^{N} w_n\cdot\max\left\{0,\theta^*(n;H_n)-d_n\right\} \tag{32}$$

is the exactly minimal total weighted tardiness for those N jobs. Generally speaking, the problem of minimizing sum (17) by (18) — (26) and constraints (27) — (30) can have multiple solutions (multiple sets of the optimal decision variables), so multiple optimal schedules ensuring the same minimal total weighted tardiness (32) can exist.

A pattern of generating instances of the job scheduling problem

Different job lengths are randomly generated by drawing numbers out of the uniform distribution [1]:

$$H_n=\psi(4\upsilon+2) \text{ for } n=\overline{1,N} \tag{33}$$

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 5 [11, 12]. The priority weight of job n is randomly generated in the identical way:

$$w_n=\psi(100\upsilon+1) \text{ for } n=\overline{1,N}. \tag{34}$$

When job lengths (33), priority weights (34), and due date shifts (9) by some N are generated for the ascending job order input so that inequality (10) holds and at least one of the inequalities in (11)

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 (4), and due dates (13) is computed by minimizing sum (17) by (18) — (26) and constraints (27) — (30). Alternatively, a descending order schedule by job lengths

$$\{H_n\}_{n=1}^{N} \text{ after } H_j^{(\text{obs})} = H_j \quad \forall\, j = \overline{1,\, N} \text{ and } H_n = H_{N-n+1}^{(\text{obs})} \text{ for } n = \overline{1,\, N}, \tag{35}$$

priority weights

$$\{w_n\}_{n=1}^{N} \text{ after } w_j^{(\text{obs})} = w_j \quad \forall\, j = \overline{1,\, N} \text{ and } w_n = w_{N-n+1}^{(\text{obs})} \text{ for } n = \overline{1,\, N}, \tag{36}$$

release dates (5), and due dates (14) is computed as well, where job lengths (35), priority weights (36), release dates (5), and due dates (14) 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 (4), and due dates (13) for the ascending job order input.

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 $\delta_{Asc}(N, c)$ and $\delta_{Desc}(N, c)$ in seconds, respectively. Each of these amounts implies computation time spent on just searching the solution to the problem of minimizing sum (17) by (18) — (26) and constraints (27) — (30), i. e. on exploring nodes by the branch-and-bound algorithm [6]. At that, the time spent on forming constraints (27) — (30) is not counted in $\delta_{Asc}(N, c)$ and $\delta_{Desc}(N, c)$. Therefore, these amounts in article [5] were called inner computation times. However, article [5] showed that difference between inner computation times and outer computation times (which count also time spent on forming the constraints) is negligible [6].

If the total number of the instances is C, then the averaged inner computation times are

$$\delta_{Asc}(N) = \frac{1}{C}\sum_{c=1}^{C}\delta_{Asc}(N, c) \tag{37}$$

and

$$\delta_{Desc}(N) = \frac{1}{C}\sum_{c=1}^{C}\delta_{Desc}(N, c). \tag{38}$$

In percentage terms, the relative difference between inner computation times (37) and (38) is

$$\mu(N) = 100 \cdot \frac{\delta_{Asc}(N) - \delta_{Desc}(N)}{\delta_{Asc}(N)}. \tag{39}$$

Relative difference (39) will be estimated by $N = \overline{2,\, 8}$ for $C \geqslant 250$ [6].

Computational study

The computational study is executed on CPU Intel Core i5-7200U@2.50 GHz using MATLAB R2018a. Relative difference (39) between inner computation times (37) and (38) for $C = 250$ is shown in Figure 1, where the horizontal zero level line is imposed. This line allows seeing where the schedule is computed faster by the respective job order input [6].

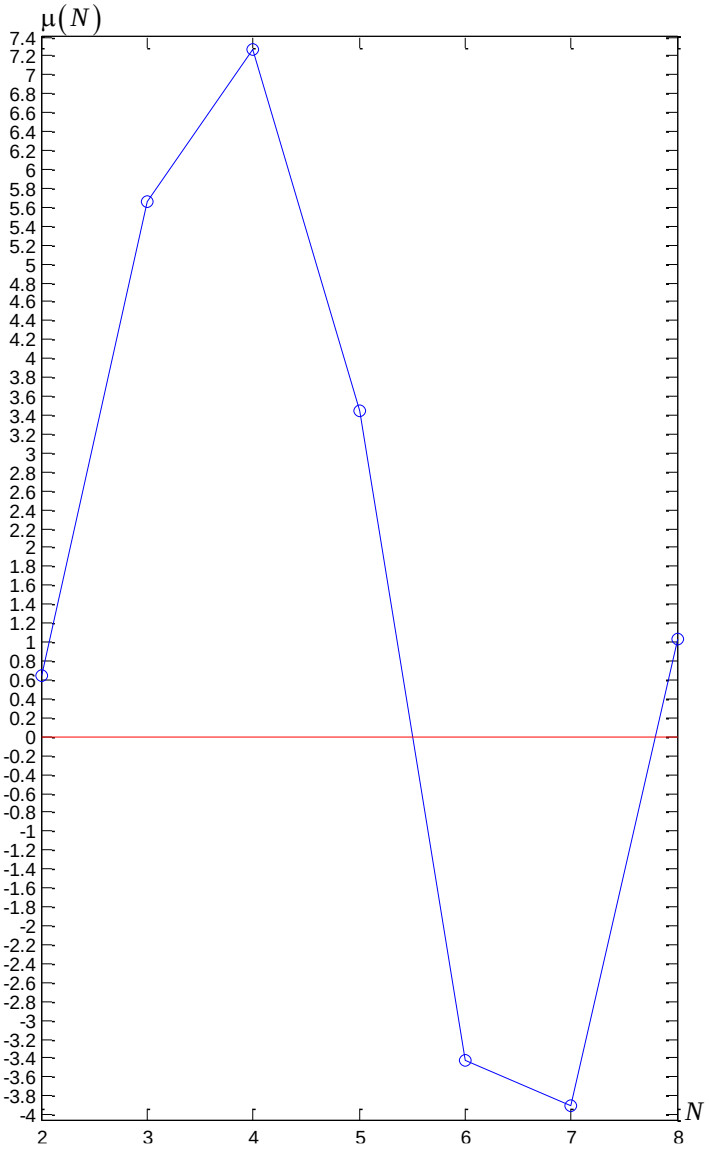

Figure 1 — Relative difference (39) between inner computation times (37) and (38) for 250 instances generated at each number of jobs to be scheduled

Unfortunately, Figure 1 cannot ensure the seeming primal conclusion about the descending job order input is faster by scheduling 2 to 5 jobs and is slower by scheduling 6 or 7 jobs. The reason is too vast scattering of the inner computation times

$$\mu(N,c)=100\cdot\frac{\delta_{Asc}(N,c)-\delta_{Desc}(N,c)}{\delta_{Asc}(N,c)} \tag{40}$$

for many instance tags $c\in\{\overline{1,\,250}\}$. Indeed, repetition of the computations for 1000 instances gives another, dissimilar to Figure 1, view of relative difference (39) presented in Figure 2 as a circle-dotted polyline. Moreover, the second version of generating 1000 instances gives the third view (the square-dotted polyline in Figure 2).

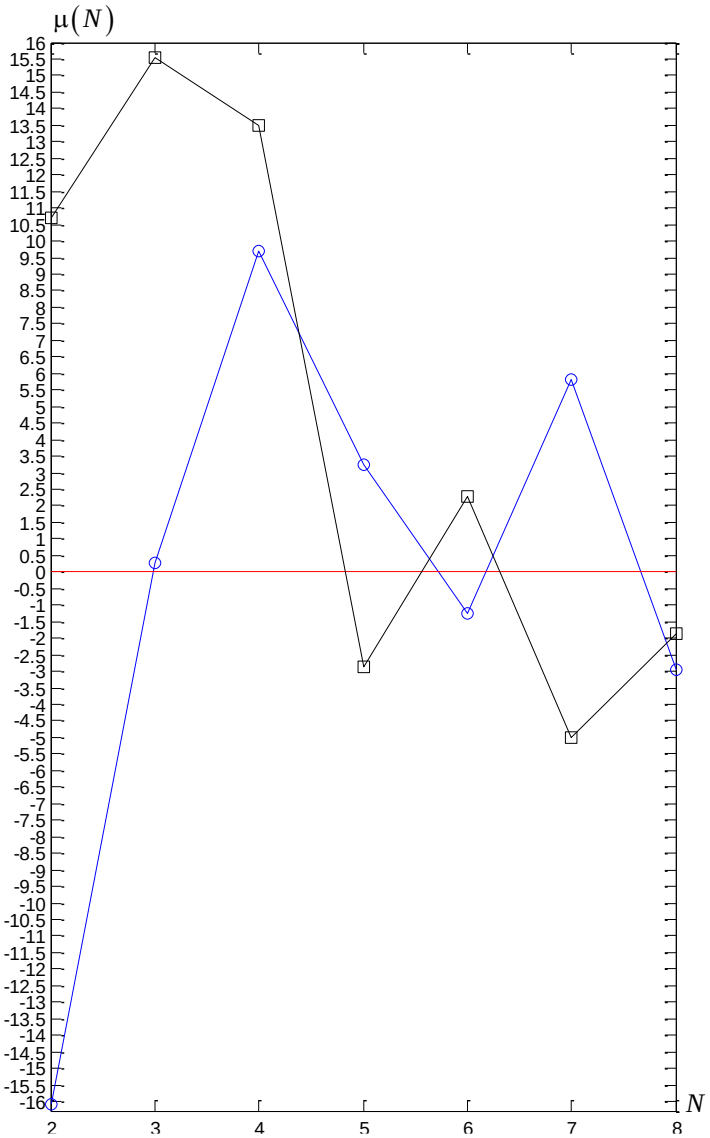

Figure 2 — Two versions of relative difference (39) between inner computation times (37) and (38) for 1000 instances (circle-dotted and square-dotted polylines) generated at each number of jobs to be scheduled

In this connection, it is useful to see how the upper and lower bounds of single-instance relative difference (40) are situated with regard to the relative difference averaged over three relative differences (39) whose polylines are plotted in Figures 1 and 2. Figure 3 shows the averaged polyline for the relative difference along with the upper bounds

$$\max_{c=1,\,C} \mu(N, c) \tag{41}$$

for the three respective versions described above. It is clearly seen that the maximal acceleration by the descending job order input achieves 50 % to 97 % according to relative difference (39), which means (by re-transforming the relative difference formula) that the descending job order input occurs to be faster by 2 to almost 33 times. An example of such a huge acceleration benefit by the descending job order input has been registered in scheduling 4 jobs, where

$$\mathbf{H} = [4 \quad 5 \quad 2 \quad 5], \quad \mathbf{W} = [64 \quad 5 \quad 20 \quad 59], \quad \mathbf{D} = [15 \quad 14 \quad 3 \quad 11]$$

and an optimal schedule is

$$\mathbf{S}^* = \left[s_t^* \right]_{1\times 16} = \begin{bmatrix} 4 & 3 & 3 & 4 & 2 & 4 & 4 & 2 & 2 & 2 & 4 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

by $\vartheta^*(4) = 10$ for the descending job order input ($r_1 = 4$), and the respective computation times have been 375 milliseconds and 11.5 milliseconds. On average, scheduling 4 jobs is about 10 % faster by the descending job order input, which is about acceleration in 42.7 milliseconds.

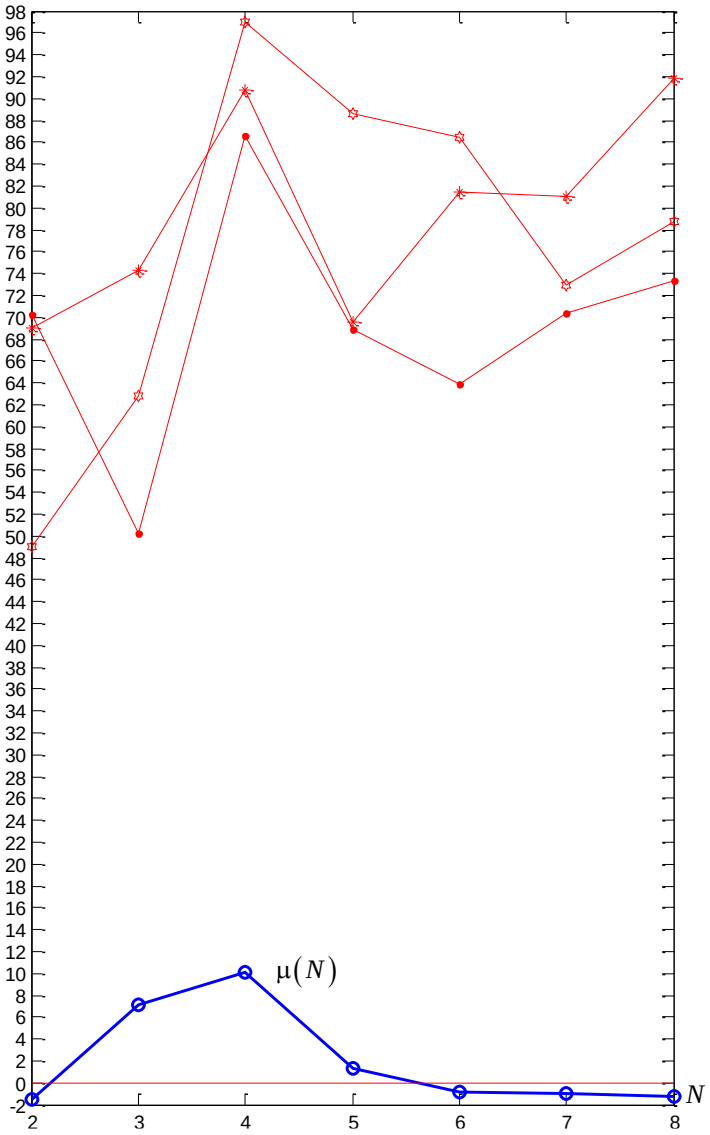

Figure 3 — The relative difference averaged over three relative differences (39) whose polylines are plotted in Figures 1 and 2, and the respective three polylines (dotted, asterisk-dotted, and hexagram-dotted ones) which are upper bounds (41)

The lower bounds

$$\min_{c=1,\,C} \mu(N, c) \tag{42}$$

for the three respective versions are shown in Figure 4. It is clearly seen that the maximal acceleration by the ascending job order input achieves 100 % to 1600 %, which directly means that

the ascending job order input occurs to be faster by 2 to almost 17 times (it is worth to remember that the relative difference or its bounds, with respect to the horizontal zero level line do not show the acceleration symmetrically). An example of such a huge acceleration benefit by the ascending job order input has been registered in scheduling 5 jobs, where

$$\mathbf{H} = \begin{bmatrix} 3 & 5 & 4 & 4 & 4 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} 86 & 16 & 8 & 5 & 11 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 5 & 12 & 12 & 10 & 13 \end{bmatrix}$$

and an optimal schedule is

$$\mathbf{S}^* = \left[s_t^* \right]_{1\times 20} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 5 & 3 & 5 & 2 & 2 & 5 & 5 & 3 & 3 & 3 & 4 & 4 & 4 & 4 \end{bmatrix}$$

by $\vartheta^*(5) = 82$ for the ascending job order input ($r_1 = 1$), and the respective computation times have been 2.7508 seconds and 46.7545 seconds.

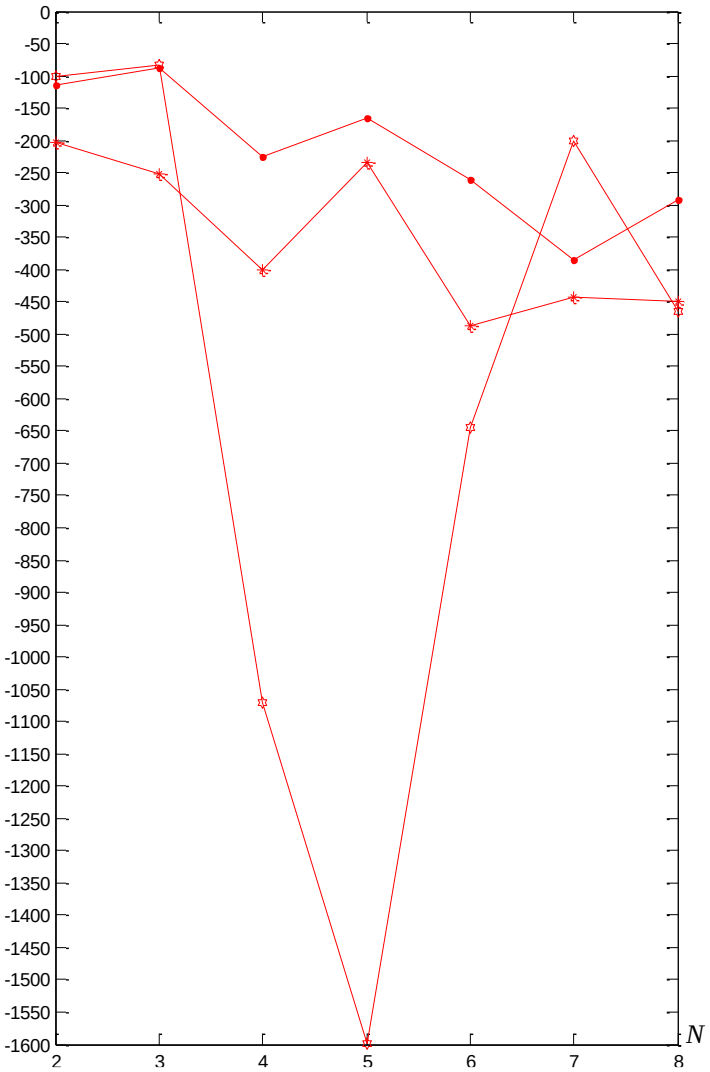

Figure 4 — The respective three polylines (dotted, asterisk-dotted, and hexagram-dotted ones) which are lower bounds (42)

The average upper bound by (41) is about 75 %, which means that the descending job order input, at its maximal acceleration instances, occurs to be 4 times faster. The average lower bound by (42) is about –389 %, which means that the ascending job order input, at its maximal acceleration

instances, occurs to be 4.89 times faster. However, this does not mean that the skew to the ascending job order input is significant as there are a few of computational artifacts (in every of those three versions of the job scheduling problem instance generations) with huge gaps between the orders.

Discussion

The main explanation for that gigantic gap between lower and upper bounds is the uniform randomness [13] of job processing periods and priority weights. Indeed, the only "carrier" of the job order input feature, apart from the release dates, is the due dates. But even here, the due dates are randomized by adding due date shifts (9) associated with job processing periods. Consequently, those intense deviations from the averaged relative difference (Figure 3) are logic results of deviations from the respective job order input feature. For instance, in scheduling 4 jobs with

$$\mathbf{H}=\begin{bmatrix}5 & 3 & 5 & 5\end{bmatrix},\ \mathbf{W}=\begin{bmatrix}39 & 57 & 8 & 6\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}5 & 6 & 7 & 20\end{bmatrix} \tag{43}$$

by

$$\mathbf{R}=\begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix},$$

an optimal schedule

$$\mathbf{S}^{*}=\left[s_t^{*}\right]_{1\times 18}=\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 4\end{bmatrix}$$

by $\vartheta^{*}(4)=162$ is found in 1.7899 seconds, whereas the descending job order input, which is

$$\mathbf{H}=\begin{bmatrix}5 & 5 & 3 & 5\end{bmatrix},\ \mathbf{W}=\begin{bmatrix}6 & 8 & 57 & 39\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}20 & 7 & 6 & 5\end{bmatrix} \tag{44}$$

by

$$\mathbf{R}=\begin{bmatrix}4 & 3 & 2 & 1\end{bmatrix},$$

takes 10.6626 seconds for its optimal schedule

$$\mathbf{S}^{*}=\left[s_t^{*}\right]_{1\times 18}=\begin{bmatrix}4 & 4 & 4 & 4 & 4 & 3 & 3 & 3 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1\end{bmatrix}.$$

The due dates in (43) are given in ascending order, but the priority weights in (43) resemble rather a descending order. This is why the difference between the computation times here does not comply with that $\mu(4)$ in Figure 3. If to set the weights in reverse order, then the ascending job order input

$$\mathbf{H}=\begin{bmatrix}5 & 3 & 5 & 5\end{bmatrix},\ \mathbf{W}=\begin{bmatrix}6 & 8 & 57 & 39\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}5 & 6 & 7 & 20\end{bmatrix} \tag{45}$$

becomes more resembling the ascending order, as well as the descending job order input

$$\mathbf{H}=\begin{bmatrix}5 & 5 & 3 & 5\end{bmatrix},\ \mathbf{W}=\begin{bmatrix}39 & 57 & 8 & 6\end{bmatrix},\ \mathbf{D}=\begin{bmatrix}20 & 7 & 6 & 5\end{bmatrix} \tag{46}$$

becomes more resembling the descending order. In this case, an optimal schedule

$$\mathbf{S}^* = \left[s_t^* \right]_{1\times 18} = \left[1 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 2 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4\right]$$

by $\vartheta^*(4) = 72$ is found in 1.8524seconds, whereas the respective optimal schedule

$$\mathbf{S}^* = \left[s_t^* \right]_{1\times 18} = \left[4 \quad 3 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\right]$$

by the descending job order input is found in 0.5923 seconds. Thus, as inputs (45) closely resemble the ascending order, and, vice versa, inputs (46) closely resemble the descending order, the difference between the computation times here complies with that $\mu(4)$ in Figure 3. Other similar examples confirm this hypothesis.

Conclusions

In tight-tardy progressive single machine scheduling with idling-free preemptions, the job order influences the speed of computing the exact solution but it is hardly possible to predict it. The matter is that the jobs can have both different lengths and different priority weights, that additionally "dithers" the respective job order input leaving only release dates which are always given in that order. It has been revealed that scheduling 3 or 4 jobs is executed on average faster by the descending job order input. However, the expected acceleration by the descending job order input cannot be estimated with a great confidence factor. This result disproves a possibility to manipulate the job order for obtaining schedules more efficiently in a single job scheduling problem or in a few such problems, just like it was for the case of equal priority weights (although in a more "light" statements, without rigor affirmations). Only if the job scheduling problems by 3 or 4 jobs resemble the descending job order input, and there are thousands of such problems, then it is recommended to use that job order input.

Considering the three classes of job scheduling problems (total tardiness by equal job lengths [5], total tardiness by different job lengths [6], and total weighted tardiness), it is obvious that the possibility to manipulate the job order for obtaining schedules more efficiently decreases as more differences in the problem parameters are included. Moreover, if even the possibility exists, it is treated as on average only. As in the two cases of equal priority weights, the descending job order input appears theoretically efficient for total weighted tardiness exact minimization also. Inputting jobs efficiently by this job order style is still possible when different job lengths and different priority weights are both scattered not much, that would be closer to the case of total tardiness by equal job lengths [5] (and equal priority weights). This case is the most promising one, where the descending job order computation time can be shorter up to 10 % and more in scheduling 2 to 6 jobs divided into two to four or even five parts each. If the job lengths and priority weights are pretty scattered, not resembling the descending job order style, the input is recommended to be made in any handy way.

REFERENCES:

1. Romanuke V. V. "Minimal total weighted tardiness in tight-tardy single machine preemptive idling-free scheduling." Applied Computer Systems, vol. 24, no. 2, 2019, pp. 150 — 160.
2. Ku W.-Y., Beck J. C. "Mixed Integer Programming models for job shop scheduling: A computational analysis." Computers & Operations Research, vol. 73, 2016, pp. 165 — 173.
3. Romanuke V. V. "A heuristic's job order gain for total weighted completion time minimization in the preemptive scheduling problem by subsequent length-equal job importance growth." Visnyk of the Lviv University. Series Appl. Math. and Informatics, iss. 27, 2019, pp. 108 — 117.
4. Romanuke V. V. "A heuristic's job order gain in pyramidal preemptive job scheduling problems for total weighted completion time minimization." Information Technology and Management Science, vol. 22, 2019, pp. 1 — 8.
5. Romanuke V. V. "Efficient exact minimization of total tardiness in tight-tardy progressive single

machine scheduling with idling-free preemptions of equal-length jobs." KPI Science News, no. 1, 2020, pp. 27 — 39.
6. Romanuke V. V. "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, 2020, pp. 19 — 36.
7. Pinedo M. L. Planning and Scheduling in Manufacturing and Services. Springer-Verlag New York, 2009: 536 p.
8. Rupanetti D., Salamy H. "Task allocation, migration and scheduling for energy-efficient real-time multiprocessor architectures." Journal of Systems Architecture, vol. 98, 2019, pp. 17 — 26.
9. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. "Optimization and approximation in deterministic sequencing and scheduling: A survey." Annals of Discrete Mathematics, no. 5, 1979, pp. 287 — 326.
10. Batsyn M., Goldengorin B., Pardalos P., Sukhov P. "Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time." Optimization Methods & Software, vol. 29, no. 5, 2014, pp. 955 — 963.
11. Romanuke V. V. "Heuristic's job order efficiency in tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs." KPI Science News, no. 2, 2020, pp. 64 — 73.
12. Romanuke V. V. "Tight-tardy progressive idling-free 1-machine preemptive scheduling by heuristic's efficient job order input." KPI Science News, no. 3, 2020, pp. 32 — 42.
13. Kneusel R. Random Numbers and Computers. Springer International Publishing, 2018: 260 p.

ЛІТЕРАТУРА:

1. Romanuke V. V. Minimal total weighted tardiness in tight-tardy single machine preemptive idling-free scheduling / V. V. Romanuke // Applied Computer Systems. — 2019. — Vol. 24, No. 2. — P. 150 — 160.
2. Ku W.-Y. Mixed Integer Programming models for job shop scheduling: A computational analysis / W.-Y. Ku, J. C. Beck // Computers & Operations Research. — 2016. — Vol. 73. — P. 165 — 173.
3. Romanuke V. V. A heuristic's job order gain for total weighted completion time minimization in the preemptive scheduling problem by subsequent length-equal job importance growth / V. V. Romanuke // Visnyk of the Lviv University. Series Appl. Math. and Informatics. — 2019. — Issue 27. — P. 108 — 117.
4. Romanuke V. V. A heuristic's job order gain in pyramidal preemptive job scheduling problems for total weighted completion time minimization / V. V. Romanuke // Information Technology and Management Science. — 2019. — Vol. 22. — P. 1 — 8.
5. Romanuke V. V. Efficient exact minimization of total tardiness in tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs / V. V. Romanuke // KPI Science News. — 2020. — No. 1. — P. 27 — 39.
6. Romanuke V. V. Job order input for efficient exact minimization of total tardiness in tight-tardy progressive single machine scheduling with idling-free preemptions / V. V. Romanuke // Scientific Papers of O. S. Popov Odesa National Academy of Telecommunications. — 2020. — No. 1. — P. 19 — 36.
7. Pinedo M. L. Planning and Scheduling in Manufacturing and Services. — Springer-Verlag New York, 2009. — 536 p.
8. Rupanetti D. Task allocation, migration and scheduling for energy-efficient real-time multiprocessor architectures / D. Rupanetti, H. Salamy // Journal of Systems Architecture. — 2019. — Vol. 98. — P. 17 — 26.
9. Graham R. L. Optimization and approximation in deterministic sequencing and scheduling: A survey / R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan // Annals of Discrete Mathematics. — 1979.. — No. 5. — P. 287 — 326.
10. Batsyn M. Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time / M. Batsyn, B. Goldengorin, P. Pardalos, P. Sukhov // Optimization Methods & Software. — 2014. — Vol. 29, No. 5. — P. 955 — 963.
11. Romanuke V. V. Heuristic's job order efficiency in tight-tardy progressive idling-free 1-machine preemptive scheduling of equal-length jobs / V. V. Romanuke // KPI Science News. — 2020. — No. 2. — P. 64 — 73.
12. Romanuke V. V. Tight-tardy progressive idling-free 1-machine preemptive scheduling by heuristic's efficient job order input / V. V. Romanuke // KPI Science News. — 2020. — No. 3. — P. 32 — 42.
13. Kneusel R. Random Numbers and Computers. — Springer International Publishing, 2018. — 260 p.