DOI: 10.20535/kpi-sn.2020.1.180877
UDC 519.161+519.852+519.687.1

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

EFFICIENT EXACT MINIMIZATION OF TOTAL TARDINESS IN TIGHT-TARDY PROGRESSIVE SINGLE MACHINE SCHEDULING WITH IDLING-FREE PREEMPTIONS OF EQUAL-LENGTH JOBS

Background. A schedule ensuring the exactly minimal total 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.
Objective. The goal is to ascertain whether the job order in tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs influences the speed of computing the exact solution. The Boolean linear programming model provided for finding schedules with the minimal total tardiness is used.
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 exact model, are excluded.
Results. Based on the three-dimensional barred plot of the relative difference between the averaged computation times, it has been shown that a possibility exists to find schedules more efficiently by manipulating the job order. For instance, schedules of 5 jobs consisting of two processing periods each are found on average by 14.67 % faster for the descending job order. In another example of 7 three-parted jobs, an optimal schedule is found on average in 69.51 seconds by the ascending job order, whereas the descending job order takes just 36.52 seconds to find it, saving thus 32.99 seconds.
Conclusions. 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.

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

Introduction

Minimization of total tardiness is a partial case of a more general problem, where jobs are associated with weights and thus total weighted tardiness is minimized [1, 2]. This partial case has a wide range of contributions and applications also because not always the job has its priority [3]. For example, if an airport supports only ordinary flights, then the only priority among arrivals comes out from their approach landing times. Then the landing schedule is worked out by these times. However, if the approach landing times appear to be close, the corresponding flights must be scheduled to land so that the delays would be minimal. At that, the runways are charged approximately with the same number of flights (both arrivals and departures) per hour (or a longer time period). Besides, the runway is charged so that there would be no idle periods (considering time measurement standards accepted in aviation, a few minutes of no taxiing to takeoff or no taxiing from landing for the runway are not counted as idling). Therefore, tight-tardy progressive single machine scheduling [4] with idling-free preemptions of equal-length jobs is a problem of the great practical importance and impact.

A schedule ensuring the exactly minimal total tardiness can be found with the respective integer linear programming problem. Models based on the branch-and-bound approach are commonly used for that [5, 6]. For tight-tardy progressive single machine scheduling, where 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), the exact model simplifies owing to no weights are included and each job has the same number of processing periods [7]. An open question is whether the exact schedule computation time changes if the release dates are input to the model in reverse order. The matter is that it was shown in article [8] that, under some additional conditions, after inputting the release dates in descending (i. e., in reverse) order the exact solution for total weighted completion time minimization is computed on average faster, than after inputting the release dates in ascending (i. e., starting from 1) order. Does the similar reversion accelerate tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs? Both positive and negative answers with their peculiarities would be useful to substantiate and maintain computations efficient.

Problem statement

The goal is to ascertain whether the job order in tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs influences the speed of computing the exact solution. The Boolean linear programming model provided for finding schedules with the minimal total 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. Then the relative difference between the computation times is to be treated. The research result is expected to either reveal or disprove a possibility to manipulate the job order for obtaining schedules more efficiently.

Minimal total tardiness by equal-length jobs

Let N be a number of jobs, $N \in \mathbb{N} \setminus \{1\}$, where job n is divided into H equal parts (i. e., has a processing period H), has a release date r_n, and a due date d_n, $n = \overline{1, N}$. Integer r_n is the time moment, at which job n becomes available for processing. So, in the case of equal-length jobs,

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

is a vector of processing periods,

$$\mathbf{R} = [r_n]_{1 \times N} \in \mathbb{N}^N \tag{2}$$

is a vector of release dates, and

$$\mathbf{D} = [d_n]_{1 \times N} \in \mathbb{N}^N \tag{3}$$

is a vector of due dates. Narrowing the problem to the already mentioned tight-tardy progressive single machine scheduling with idling-free preemptions, the release dates given in ascending order are

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

and the release dates given in descending order are

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

The due dates are tightly set after the release dates, in whichever order they are given:

$$d_n = r_n + H - 1 + b_n \quad \forall n = \overline{1, N} \tag{6}$$

for ascending order and

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

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

$$\mathbf{B} = [b_n]_{1 \times N} = \psi(H \cdot \Xi(1, N)) \tag{8}$$

with operator $\Xi(1, N)$ returning a pseudorandom $1 \times N$ vector whose entries are drawn from the standard normal distribution (with zero mean and unit variance), and function $\psi(\xi)$ returning the integer part of number ξ (e. g., see [4, 5, 9]). While components of due date shift vector (8) are returned in a (-1)-non-descending order, i. e. while condition

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

is true, vector (8) is re-generated. In other words, components of due date shift vector (8) are not given in the (-1)-non-descending order. Otherwise, operator $\Xi(1, N)$ is executed again for re-generating vector (8). For instance, vector

$$\mathbf{B} = [1 \quad 0 \quad 2 \quad 2 \quad 3]$$

does not fit here, whereas vector

$$\mathbf{B} = [1 \quad -1 \quad 2 \quad 2 \quad 3]$$

fits. Besides, vector (8) is re-generated if

$$d_{n_1} < 1 \quad \text{for some} \quad n_1 \in \{\overline{1, N}\}. \tag{10}$$

So, for a properly given due date shift vector (8), where both conditions (9) and (10) must be violated, due dates (6) set in the order corresponding to ascending order of the release dates (4) are

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

and due dates (7) set in the order corresponding to descending order of the release dates (5) are

$$d_n = N + H - n + b_{N-n+1} \quad \forall n = \overline{1, N}. \tag{12}$$

Thus, components of due dates vector (3) are neither given in non-descending order for the case of the ascending job order by (4), nor are given in non-ascending order for the case of the descending job order by (5). This is done so because in the case of when either inequalities

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

or

$$d_n \ge d_{n+1} \quad \forall\, n = \overline{1,\, N-1} \tag{14}$$

are true, a schedule ensuring the exactly minimal total tardiness is found trivially, without resorting to any algorithm or model. This fact is going to be proved below.

The goal is to minimize the total tardiness, i. e. to schedule N jobs so that sum

$$\sum_{n=1}^{N} \max\{0,\, \theta(n;\, H) - d_n\} \tag{15}$$

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

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

This goal is equivalent to minimizing sum

$$\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{t=1}^{N\cdot H} \lambda_{nht} x_{nht} \tag{16}$$

by the known Boolean linear programming model (applied for minimizing total weighted completion time also) [5, 8], 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)

$$\{\{\{\lambda_{nht}\}_{t=1}^{N\cdot H}\}_{h=1}^{H}\}_{n=1}^{N}$$

are calculated as follows:

$$\lambda_{nht} = 0 \tag{17}$$

by

$$r_n - 1 + h \le t \le (N-1)H + h \quad \forall\, h = \overline{1,\, H-1} \tag{18}$$

and

$$\lambda_{nht} = \alpha \tag{19}$$

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 (18) is not true;

$$\lambda_{nHt} = 0 \tag{20}$$

by

$$r_n - 1 + H \le t \le d_n \tag{21}$$

and

$$\lambda_{nHt} = t - d_n \tag{22}$$

by

$$d_n < t \le N\cdot H \tag{23}$$

and

$$\lambda_{nHt} = \alpha \tag{24}$$

when both (21) and (23) are not true. In (19) and (24), for instance,

$$\alpha = \sum_{n=1}^{N}\sum_{t=1}^{N\cdot H} t = \frac{N^2\cdot H\cdot(N\cdot H+1)}{2} \tag{25}$$

can be used [4, 5, 8]. So, sum (16) is defined on set

$$X = \{\{\{x_{nht}\}_{n=1}^{N}\}_{h=1}^{H}\}_{t=1}^{N\cdot H} \in \mathscr{X}_{0-1} \subset \mathbb{R}^{(NH)^2},$$

which is an $N\times H\times(N\cdot H)$ matrix of ones and zeros, where $\mathscr{X}_{0-1}$ is a set of all possible such matrices. It is an integer binary lattice in $\mathbb{R}^{(NH)^2}$ whose vertices consist of only ones and zeros. Consequently, the goal is to find such a set

$$X^* = \{\{\{x^*_{nht}\}_{n=1}^{N}\}_{h=1}^{H}\}_{t=1}^{N\cdot H} \in \mathscr{X}_{[0-1]} \subset \mathscr{X}_{0-1} \tag{26}$$

on which minimum

$$\min_{X\in\mathscr{X}_{[0-1]}\subset\mathscr{X}_{0-1}} \sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{t=1}^{N\cdot H} \lambda_{nht} x_{nht} \tag{27}$$

is achieved by constraints constituting integer binary lattice $\mathscr{X}_{[0-1]}$ [4, 5, 8]:

$$x_{nht} \in \{0,\, 1\} \text{ by } n = \overline{1,\, N} \text{ and } h = \overline{1,\, H} \text{ and } t = \overline{1,\, N\cdot H}, \tag{28}$$

$$\sum_{t=1}^{N\cdot H} x_{nht} = 1 \text{ by } n = \overline{1,\, N} \text{ and } h = \overline{1,\, H}, \tag{29}$$

$$\sum_{n=1}^{N}\sum_{h=1}^{H} x_{nht} = 1 \text{ by } t = \overline{1,\, N\cdot H}, \tag{30}$$

$$\sum_{j=t+1}^{N\cdot H}\sum_{h=1}^{H-1} x_{nhj} + Hx_{nHt} \le H \text{ by } n = \overline{1,\, N} \text{ and } t = \overline{1,\, N\cdot H-1}. \tag{31}$$

If (26) is a solution of the problem, it is the optimal job schedule

$$\mathbf{S}^* = [s_t^*]_{1\times(N\cdot H)} \text{ by } s_t^* \in \{\overline{1, N}\} \text{ for every } t = \overline{1, N\cdot H}. \tag{32}$$

In schedule (32),

$$s^*_{\theta^*(n;\,h)} = n \quad \forall h = \overline{1, H} \text{ by } \theta^*(n;\,h) \in \{\overline{1, N\cdot H}\}$$
$$\text{and } \theta^*(n;\,h) < \theta^*(n;\,h+1) \text{ for } h = \overline{1, H-1}.$$

Thus, $\theta^*(n;\,H)$ is a moment after which job n is completed, and, according to sum (15),

$$\vartheta^*(N, H) = \sum_{n=1}^{N} \max\{0, \theta^*(n;\,H) - d_n\} \tag{33}$$

is the exactly minimal total tardiness for those N jobs. Alternatively, amount (33) can be found by solution (26):

$$\vartheta^*(N, H) = \sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{t=1}^{N\cdot H} \lambda_{nht} x^*_{nht}\,. \tag{34}$$

Obviously, a few optimal schedules ensuring the same minimum (33) or, if calculating straightforwardly within model (17)–(31), minimum (34), can exist. For example, a problem of scheduling 4 jobs divided into three parts each (i. e., $N = 4$, $H = 3$) has release dates (in ascending order)

$$\mathbf{R} = [r_n]_{1\times4} = [1 \;\; 2 \;\; 3 \;\; 4] \tag{35}$$

by (4) and due dates (in ascending order)

$$\mathbf{D} = [d_n]_{1\times4} = [3 \;\; 4 \;\; 5 \;\; 6] \tag{36}$$

by (11) with $b_n = 0 \;\; \forall n = \overline{1, 4}$. Schedule

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;2\;\;2\;\;2\;\;3\;\;3\;\;3\;\;4\;\;4\;\;4] \tag{37}$$

is optimal for

$$\mathbf{H} = [3 \;\; 3 \;\; 3 \;\; 3],$$

(35) and (36), ensuring total tardiness

$$\vartheta^*(4, 3) = \sum_{n=1}^{4} \max\{0, \theta^*(n;\,3) - d_n\}$$
$$= \max\{0, 3-3\} + \max\{0, 6-4\}$$
$$+ \max\{0, 9-5\} + \max\{0, 12-6\} = 12$$

by statement (33). However, schedule

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;3\;\;3\;\;3\;\;4\;\;4\;\;4\;\;2\;\;2\;\;2] \tag{38}$$

for release dates (35) and due dates (36) is optimal as well, ensuring the same amount of total tardiness:

$$\vartheta^*(4, 3) = \sum_{n=1}^{4} \max\{0, \theta^*(n;\,3) - d_n\}$$
$$= \max\{0, 3-3\} + \max\{0, 12-4\}$$
$$+ \max\{0, 6-5\} + \max\{0, 9-6\} = 12.$$

Note that schedules (37) and (38) differ in only that job 2 in schedule (38) "leapt" over jobs 3 and 4. Furthermore, it is easy to check that schedules

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;4\;\;4\;\;4\;\;3\;\;3\;\;3\;\;2\;\;2\;\;2], \tag{39}$$

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;4\;\;4\;\;4\;\;2\;\;2\;\;2\;\;3\;\;3\;\;3], \tag{40}$$

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;3\;\;3\;\;3\;\;2\;\;2\;\;2\;\;4\;\;4\;\;4], \tag{41}$$

$$\mathbf{S}^* = [s_t^*]_{1\times12} = [1\;\;1\;\;1\;\;2\;\;2\;\;2\;\;4\;\;4\;\;4\;\;3\;\;3\;\;3] \tag{42}$$

are optimal here also, ensuring minimal total tardiness $\vartheta^*(4, 3) = 12$.

In the considered example with due dates (36) given in ascending order, optimal schedule (37) is determined trivially, without searching for solution (26) of problem (27) by (28)–(31). Such examples and others related to it, where due dates are given in non-descending order, will not be included into the computational study. The following theorem rigorously describes a class of tight-tardy progressive single machine scheduling problems with idling-free preemptions of equal-length jobs which do not need model (17)–(31).

Theorem 1. A single machine scheduling problem with idling-free preemptions of equal-length jobs (1) having release dates (2) as (4) and due dates (3) by (13) has an optimal schedule

$$\mathbf{S}^* = [s_t^*]_{1\times(N\cdot H)} \text{ by } s_t^* = n \;\; \forall t = \overline{(n-1)H+1, nH} \text{ for } n = \overline{1, N} \tag{43}$$

whose total tardiness

$$\vartheta_{1\ldots N} = \sum_{n=1}^{N} \max\{0, nH - d_n\} \tag{44}$$

is minimal.

P r o o f. Suppose that amount (44) can be decreased by interchanging some different jobs m and p in schedule (43), where $m < p$. The interchange implies that the job is moved either forward or backward as a comprehensive whole, with all its H parts standing in a row in schedule (43). Firstly, let

$$mH - d_m \geq 0 \quad \text{and} \quad pH - d_p \geq 0 \tag{45}$$

for these jobs. The collective tardiness of jobs m and p in schedule (43) is

$$\max\{0,\, mH - d_m\} + \max\{0,\, pH - d_p\} \\ = mH - d_m + pH - d_p. \tag{46}$$

Then, in the new schedule, job m is completed at moment pH, and job p is completed at moment mH. As job m is completed later than in schedule (43), then its tardiness is greater than that in (43):

$$\max\{0,\, pH - d_m\} > \max\{0,\, mH - d_m\} \geq 0. \tag{47}$$

If the tardiness of job p, scheduled now earlier than in schedule (43), is

$$mH - d_p \geq 0$$

then the collective tardiness of jobs m and p in the new schedule

$$\max\{0,\, pH - d_m\} + \max\{0,\, mH - d_p\} \\ = pH - d_m + mH - d_p \tag{48}$$

is the same as the collective tardiness of these jobs in schedule (43). Otherwise, if

$$mH - d_p < 0 \tag{49}$$

then the collective tardiness of jobs m and p in the new schedule is greater than (46):

$$\max\{0,\, pH - d_m\} + \max\{0,\, mH - d_p\} \\ = pH - d_m > mH - d_m + pH - d_p. \tag{50}$$

Secondly, let

$$mH - d_m < 0 \quad \text{and} \quad pH - d_p \geq 0. \tag{51}$$

The collective tardiness of jobs m and p in schedule (43) is

$$\max\{0,\, mH - d_m\} + \max\{0,\, pH - d_p\} \\ = pH - d_p. \tag{52}$$

As $d_m \leq d_p$, inequality (49) is true and inequality

$$pH - d_m \geq pH - d_p \tag{53}$$

is true as well. Then the collective tardiness of jobs m and p in the new schedule is not less than (52):

$$\max\{0,\, pH - d_m\} + \max\{0,\, mH - d_p\} \\ = pH - d_m \geq pH - d_p. \tag{54}$$

Finally, let

$$mH - d_m \geq 0 \quad \text{and} \quad pH - d_p < 0. \tag{55}$$

The collective tardiness of jobs m and p in schedule (43) is

$$\max\{0,\, mH - d_m\} + \max\{0,\, pH - d_p\} \\ = mH - d_m. \tag{56}$$

As $d_m \leq d_p$, inequalities (49) and (53) are both true again. Then the collective tardiness of jobs m and p in the new schedule is greater than (56):

$$\max\{0,\, pH - d_m\} + \max\{0,\, mH - d_p\} \\ = pH - d_m > mH - d_m. \tag{57}$$

Therefore, in each of the cases (45), (51), (55), total tardiness (44) is not decreased. If, occasionally,

$$mH - d_m < 0 \quad \text{and} \quad pH - d_p < 0, \tag{58}$$

then the collective tardiness of jobs m and p in schedule (43) is zero, and thus it cannot be decreased. Interchanging a pair of the completing parts of two jobs leads to the same conclusions by straightforwardly using (45)–(58). In general, these statements imply that moving an earlier job forward cannot decrease total tardiness (44). Consequently, schedule (43) is optimal and thus total tardiness (44) is minimal. The theorem has been proved.

It is worth to note that the conditions of Theorem 1 hold for any number of jobs. The case with due dates (3) by (14) is easily proved by using the obvious symmetry in reasoning. In a partial case of Theorem 1, when due dates themselves are given in ascending order as

$$d_n = H + n - 1 \quad \forall\, n = \overline{1, N} \tag{59}$$

(i. e., $b_n = 0 \;\; \forall\, n = \overline{1, N}$), total tardiness (44) is

$$\vartheta_{1 \ldots N} = \sum_{n=1}^{N} \max\{0,\, nH - d_n\} \\ = \sum_{n=1}^{N} \max\{0,\, (n-1)(H-1)\}$$

$$= \sum_{n=2}^{N} (n-1)(H-1) = \frac{N(N-1)(H-1)}{2}. \qquad (60)$$

The optimal schedules for this partial case possess a property owing to which they can be derived from schedule (43) just as schedules (38)–(42) are derived from schedule (37).

Theorem 2. In a single machine scheduling problem with idling-free preemptions of equal-length jobs (1) having release dates (2) as (4) and due dates (59), where schedule (43) is optimal, for any permutation set

$$\begin{gathered} M = \{m_{n-1}\}_{n=2}^{N} \subset \{\overline{2,\ N}\} \text{ by} \\ M \cap \{\overline{2,\ N}\} = \{\overline{2,\ N}\} \end{gathered} \qquad (61)$$

schedule

$$\begin{gathered} \mathbf{S}^* = [s_t^*]_{1 \times (N \cdot H)} \text{ by } s_t^* = 1 \quad \forall t = \overline{1,\ H} \text{ and} \\ \text{by } s_t^* = m_{n-1} \quad \forall t = \overline{(n-1)H + 1,\ nH} \\ \text{for } n = \overline{2,\ N} \end{gathered} \qquad (62)$$

is optimal as well by $H \geq N - 1$.

P r o o f. Consider interchanging some different jobs m and p in schedule (43), where $1 < m < p$ (as previously, the interchange implies that the job is moved either forward or backward as a comprehensive whole). The collective tardiness of jobs m and p in schedule (43) is

$$\begin{aligned} &\max\{0,\ mH - d_m\} + \max\{0,\ pH - d_p\} \\ &= \max\{0,\ mH - (H + m - 1)\} \\ &+ \max\{0,\ pH - (H + p - 1)\} \\ &= \max\{0,\ (m-1)(H-1)\} \\ &+ \max\{0,\ (p-1)(H-1)\} \\ &= (m-1)(H-1) + (p-1)(H-1) \\ &= (m + p - 2)(H-1). \end{aligned} \qquad (63)$$

The collective tardiness of jobs m and p after the interchange is

$$\begin{aligned} &\max\{0,\ pH - d_m\} + \max\{0,\ mH - d_p\} \\ &= \max\{0,\ pH - (H + m - 1)\} \\ &+ \max\{0,\ mH - (H + p - 1)\} \\ &= \max\{0,\ (p-1)H - (m-1)\} \\ &+ \max\{0,\ (m-1)H - (p-1)\}. \end{aligned} \qquad (64)$$

As $1 < m < p$, then

$$(p-1)H - (m-1) > 0. \qquad (65)$$

The minimal value of statement

$$(m-1)H - (p-1) \qquad (66)$$

in (64) is achieved at the least m and the greatest p. So, at $m = 2$ and $p = N$ statement (66) is

$$(2-1)H - (N-1) = H - (N-1) \geq 0,$$

i. e.

$$(m-1)H - (p-1) \geq 0. \qquad (67)$$

Owing to (65) and (67), collective tardiness (64) of jobs m and p after the interchange is

$$\begin{aligned} &\max\{0,\ (p-1)H - (m-1)\} \\ &+ \max\{0,\ (m-1)H - (p-1)\} \\ &= (p-1)H - (m-1) + (m-1)H - (p-1) \\ &= (p-1)(H-1) + (m-1)(H-1) \\ &= (m + p - 2)(H-1), \end{aligned} \qquad (68)$$

i. e. it is (63). Obviously, in scheduling 3 jobs, there are only two permutation sets

$$M = \{m_{n-1}\}_{n=2}^{3} = \{3,\ 2\}$$

and

$$M = \{m_{n-1}\}_{n=2}^{3} = \{2,\ 3\}$$

and thus two schedules (62) are optimal for this case. In scheduling more than 3 jobs, interchanging only two jobs in schedule (43) does not cover permutation set (61). Suppose that some different jobs q and w are interchanged now in an optimal schedule obtained by initially interchanging only jobs m and p in schedule (43), where $1 < q < w$. The initial interchange can be fulfilled for multiple times, and it is done only for those jobs, where job m is completed at moment mH and job p is completed at moment pH. Therefore, suppose that, in the new optimal schedule, job q is completed at moment mH and job w is completed at moment pH. The collective tardiness of jobs q and w in the new schedule is

$$\begin{aligned} &\max\{0,\ mH - d_q\} + \max\{0,\ pH - d_w\} \\ &= \max\{0,\ mH - (H + q - 1)\} \\ &+ \max\{0,\ pH - (H + w - 1)\} \\ &= \max\{0,\ (m-1)H - (q-1)\} \\ &+ \max\{0,\ (p-1)H - (w-1)\}. \end{aligned} \qquad (69)$$

Owing to condition $H \geq N-1$ and the reasoning identical to deducing inequality (67), amount (69) is

$$\max\{0, (m-1)H-(q-1)\} + \max\{0, (p-1)H-(w-1)\} = (m-1)H-(q-1)+(p-1)H-(w-1) = (m+p-2)H-q-w+2. \quad (70)$$

Then the collective tardiness of jobs q and w after the interchange is the same as amount (70):

$$\max\{0, pH-d_q\} + \max\{0, mH-d_w\} = \max\{0, (p-1)H-(q-1)\} + \max\{0, (m-1)H-(w-1)\} = (p-1)H-(q-1)+(m-1)H-(w-1) = (m+p-2)H-q-w+2. \quad (71)$$

Consequently, any interchange of two jobs does not change their collective tardiness. This is equivalent to permutations in set (61), after which schedule (62) is still optimal. The theorem has been proved.

Clearly, the total tardiness for schedules (62) by (61) is (60). Theorem 2, whose conditions imply scheduling no less than 3 jobs, helps as building such schedules, as well as avoiding non-optimal schedules. For example,

$$\mathbf{S}^* = [s_t^*]_{1\times 8} = [1 \;\; 1 \;\; 2 \;\; 2 \;\; 3 \;\; 3 \;\; 4 \;\; 4] \quad (72)$$

is an optimal schedule for release dates (35) and due dates

$$\mathbf{D} = [d_n]_{1\times 4} = [2 \;\; 3 \;\; 4 \;\; 5],$$

and it ensures the minimal total tardiness

$$\vartheta^*(4, 2) = \sum_{n=1}^{4} \max\{0, \theta^*(n; 2) - d_n\} = \frac{4\cdot(4-1)\cdot(2-1)}{2} = 6,$$

by (60), but schedule

$$\mathbf{S} = [s_t]_{1\times 8} = [1 \;\; 1 \;\; 4 \;\; 4 \;\; 3 \;\; 3 \;\; 2 \;\; 2] \quad (73)$$

is not optimal according to Theorem 2 because

$$H = 2 < N-1 = 3.$$

Indeed, total tardiness of schedule (73) is greater than that of (72):

$$\sum_{n=1}^{4} \max\{0, \theta(n; 2) - d_n\} = \max\{0, 2-2\} + \max\{0, 8-3\} + \max\{0, 6-4\} + \max\{0, 4-5\} = 7,$$

so schedule (73) is not optimal.

Obviously, both Theorems 1 and 2 ensure the reflectively symmetric conclusions for the case, when the release dates and due dates are given in descending order, by straightforwardly using (45)—(58) and (63)—(71). All these cases are excluded from the computational study.

A pattern of generating instances of the job scheduling problem

Instances of the job scheduling problem will be generated by the definite numbers of jobs N and of job parts H. When due date shift vector (8) is properly generated, due dates (11) corresponding to ascending order and due dates (12) corresponding to descending order are calculated. Then an ascending order schedule by release dates (4) and due dates (11) is computed. Alternatively, a descending order schedule by release dates (5) and due dates (12) 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 $\delta_{Asc}(N, H, c)$ and $\delta_{Desc}(N, H, c)$ in seconds, respectively. Each of these amounts implies computation time spent on just searching the solution of problem (27), i. e. on exploring nodes by the branch-and-bound algorithm. At that, the time spent on forming the integer binary lattice $\mathscr{X}_{[0-1]}$ by (28)—(31) is not counted in $\delta_{Asc}(N, H, c)$ and $\delta_{Desc}(N, H, c)$. Therefore, let these amounts be called inner computation times. If the total number of the instances is C, then the averaged inner computation times are

$$\delta_{Asc}(N, H) = \frac{1}{C}\sum_{c=1}^{C} \delta_{Asc}(N, H, c) \quad (74)$$

and

$$\delta_{Desc}(N, H) = \frac{1}{C}\sum_{c=1}^{C} \delta_{Desc}(N, H, c). \quad (75)$$

In percentage terms, the relative difference between inner computation times (74) and (75) is

$$\mu_{in}(N, H) = 100 \cdot \frac{\delta_{Asc}(N, H) - \delta_{Desc}(N, H)}{\delta_{Asc}(N, H)}. \quad (76)$$

However, if to count also the time spent on forming the integer binary lattice $\mathscr{X}_{[0-1]}$ by (28)–(31), the difference between the computation times for both ascending and descending orders may be other. Thus, denote the schedule computation times by ascending order and descending order, spent on forming the integer binary lattice $\mathscr{X}_{[0-1]}$ by (28)–(31) and searching the solution of problem (27), by $\upsilon_{Asc}(N, H, c)$ and $\upsilon_{Desc}(N, H, c)$ in seconds, respectively. Let these amounts be called outer computation times. It is quite obvious that

$$\upsilon_{Asc}(N, H, c) > \delta_{Asc}(N, H, c) \quad (77)$$

and

$$\upsilon_{Desc}(N, H, c) > \delta_{Desc}(N, H, c). \quad (78)$$

The averaged outer computation times are

$$\upsilon_{Asc}(N, H) = \frac{1}{C}\sum_{c=1}^{C}\upsilon_{Asc}(N, H, c) \quad (79)$$

and

$$\upsilon_{Desc}(N, H) = \frac{1}{C}\sum_{c=1}^{C}\upsilon_{Desc}(N, H, c). \quad (80)$$

In percentage terms, the relative difference between outer computation times (79) and (80) is

$$\mu_{out}(N, H) = 100 \cdot \frac{\upsilon_{Asc}(N, H) - \upsilon_{Desc}(N, H)}{\upsilon_{Asc}(N, H)}. \quad (81)$$

Relative differences (76) and (81) will be estimated over a natural rectangular lattice, which is formed by

$$N = \overline{2, 10} \quad \text{and} \quad H = \overline{2, 6}. \quad (82)$$

It is assumed that the rectangular lattice formed by (82) is sufficient to obtain reliable statistics for unbiased estimation of relative differences (76) and (81). However, the number of instances generated for greater integers N and H, at which the exact schedule computation time grows immensely, will be less than that for a fewer jobs divided into a fewer parts.

Computational study

Regularly, it is sufficient to generate and explore 500 instances for a pair of N and H, where $N < 7$ and $H < 7$. Nevertheless, even scheduling 4 jobs divided into five parts each may take a few seconds, so it is reasonable to decrease the number of instances generated for such pairs down to 100. Further increment of either N or H and both of them leads to considerable increment of computation times. Thus, scheduling 10 jobs divided just into two parts each may take up to 10 minutes. So, $C = 30$ for this case. Starting off $N = 8$ and $H = 5$, the computation time drags on beyond 2 hours, which is technically called a timeout. The same happens at scheduling 10 jobs divided into more than two parts. Eventually, the matrix with values of number C planned over the rectangular lattice formed by (82) is

$$\begin{bmatrix} 500 & 500 & 100 & 100 & 100 & 10 & 0 & 0 & 0 \\ 500 & 500 & 100 & 100 & 100 & 10 & 0 & 0 & 0 \\ 500 & 500 & 500 & 500 & 500 & 10 & 10 & 10 & 0 \\ 500 & 500 & 500 & 500 & 500 & 50 & 50 & 10 & 0 \\ 500 & 500 & 500 & 500 & 500 & 250 & 250 & 120 & 30 \end{bmatrix} \quad (83)$$

and these values are visualized in Fig. 1. There are eight pairs of integers N and H, at which no instances are generated by reason of 2-hour timeouts. The pairs corresponding to a greater number of the generated instances are marked with circles of bigger size and lighter color.

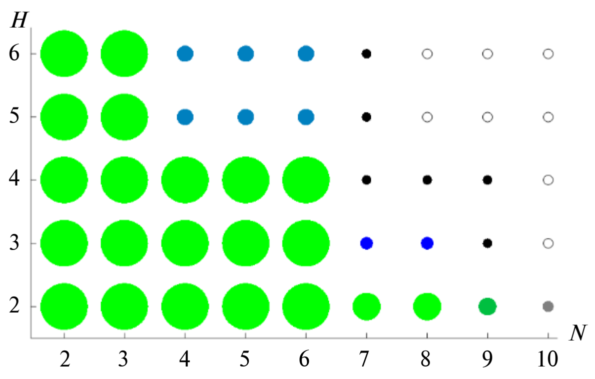

Fig. 1. The rectangular lattice, where circles differing in size and color show how many instances are generated according to matrix (83); the pairs of N and H, at which no instances are generated by reason of timeouts, are marked with empty circles

The percentage of the relative difference between outer computation times by (81) is barred in Fig. 2, where bars of negative values of the difference are marked with a lighter color. In fact, it is pretty close to the relative difference between inner computation times by (76) similarly barred in Fig. 3. Inequalities (77) and (78) are surely true, but they do not really matter here. It appears that both values

$$\mu_{out}(7, 5) \approx 59.4 \quad \text{and} \quad \mu_{in}(7, 5) \approx 59.4$$

are caused by an artifact of pseudorandomness of operator $\Xi(1, N)$ for generating due date shift vector (8). On the other hand, there are 2-hour timeouts at $N = 7$ and $H = 5$, so they additionally induce computational artifacts.

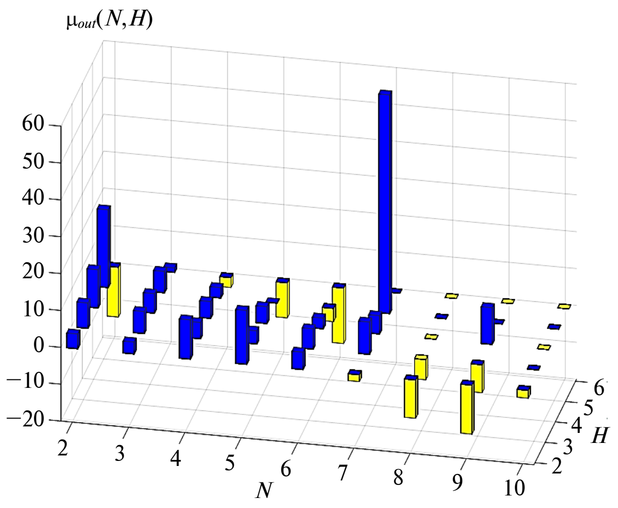

Fig. 2. The percentage of the relative difference between outer computation times

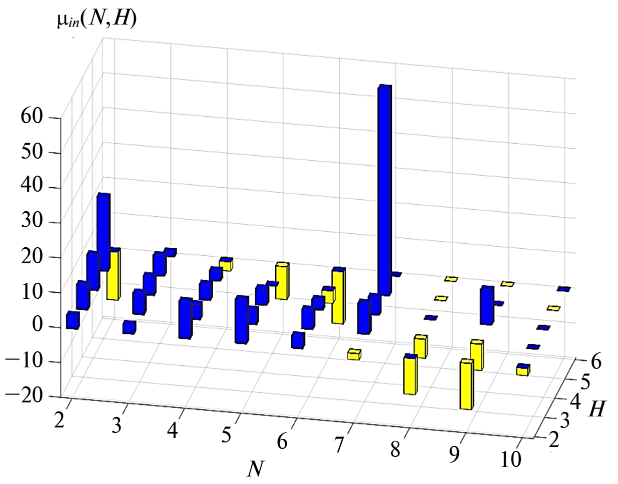

Fig. 3. The percentage of the relative difference between inner computation times

Apart from the artifact, some regularities are seen in both Figs. 2 and 3. Firstly, the descending job order has distinctly shorter computation time in scheduling 2 to 6 jobs divided into two to four parts each. Here, schedules by the descending job order have been found by 2.95 % to 14.67 % faster. Secondly, this regularity vanishes as either the number of jobs is increased or the job has a greater number of processing periods (i. e., is divided into a greater number of parts). Thus, schedules of 7 to 10 jobs consisting of just two processing periods each have been found by 1.83 % to 13.26 % faster by the ascending job order. Roughly the same advantage of the ascending job order has been revealed in scheduling 2 to 6 jobs consisting of six processing periods each.

As it has been already mentioned above, the computational study is organized so, that the conditions of Theorem 1 are excluded. Owing to this, the trivial solutions in the form of optimal schedule (43) whose total tardiness is (44) are not included into the statistics of computation times as they do not need model (17)—(31). However, the obtained statistics contain cases in which the schedule trivially coincides with optimal schedule (43), although not obeying the conditions of Theorem 1 (see it in Fig. 4). In scheduling 2 jobs, the percentage of such cases is decreasing, being naturally maximal (72.4 %) at dividing the jobs into just two parts each. Some kind of a pseudorandomness artifact can be seen in scheduling 3 to 6 jobs divided into just two parts each, where no trivial schedules have been registered except for $H = 3$ whose percentage is 0.2 %. The same weird zeros are at $N = 5$ and $N = 6$ for $H = 3$. After all, despite this weird artifact, the trivial solutions are not believed to affect the statistics, although it is worth to know about the percentage and likely unexpectednesses of how it changes.

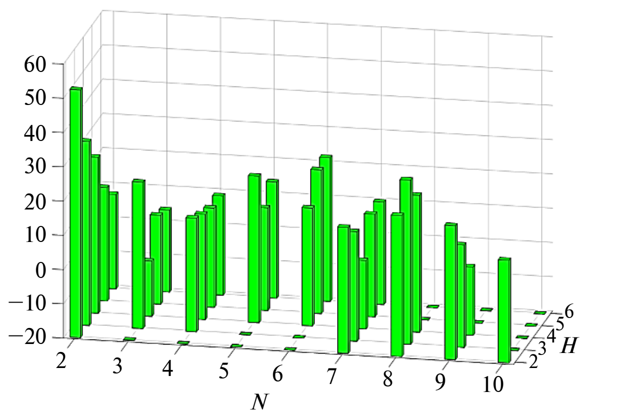

Fig. 4. The percentage of cases (over the rectangular lattice in Fig. 1) in which the schedule trivially coincides with optimal schedule (43), although not obeying the conditions of Theorem 1

Before discussing the research result, it is also worth to note that the exposed artifacts shall not be tried for rectification or something. Repetitions of schedules and their triviality like schedule (43) are not excluded in common practice. For example, scheduling arrivals and departures at airports (where the tardiness often occurs having very high costs) is tried to be as constant as possible by reason of the comfort and convenience for passengers and pilots. Arrivals and departures at railway stations are the similar example, with lesser costs of the tardiness, though. Therefore, the artifacts (related to both pseudorandomness and computations) have been stored and subsequently exposed in the barred plots. The artifacts and other seeming "weirdnesses" are a part of reality and so they should be in the research result along with the "common" statistics.

Discussion

The research result obtained on the basis of Figs. 2–4 seemingly proves a possibility to find schedules more efficiently by manipulating the job order. Thus, $\mu_{out}(5, 2) \approx 14.67$ implying that schedules of 5 jobs consisting of two processing periods each will be found on average by 14.67 % faster for the descending job order. For instance, consider the respective job scheduling problem with due dates

$$\mathbf{D} = [d_n]_{1\times 5} = [5 \quad 1 \quad 4 \quad 3 \quad 10] \tag{84}$$

for the ascending job order and

$$\mathbf{D} = [d_n]_{1\times 5} = [10 \quad 3 \quad 4 \quad 1 \quad 5] \tag{85}$$

for the descending job order. An optimal schedule for due dates (84) is

$$\mathbf{S}^* = [s_t^*]_{1\times 10} = [1 \quad 2 \quad 2 \quad 3 \quad 1 \quad 3 \quad 4 \quad 4 \quad 5 \quad 5] \tag{86}$$

whose total tardiness is

$$\vartheta^*(5, 2) = \sum_{n=1}^{5} \max\{0, \theta^*(n; 2) - d_n\}$$
$$= \max\{0, 5-5\} + \max\{0, 3-1\} + \max\{0, 6-4\} + \max\{0, 8-3\} + \max\{0, 10-10\} = 9.$$

It is worth to note that, along with optimal schedule (86), schedule

$$\mathbf{S}^* = [s_t^*]_{1\times 10} = [1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 1 \quad 4 \quad 4 \quad 5 \quad 5] \tag{87}$$

is optimal also as its total tardiness is the same:

$$\vartheta^*(5, 2) = \sum_{n=1}^{5} \max\{0, \theta^*(n; 2) - d_n\}$$
$$= \max\{0, 6-5\} + \max\{0, 3-1\} + \max\{0, 5-4\} + \max\{0, 8-3\} + \max\{0, 10-10\} = 9.$$

Either schedule (86) and schedule (87) is found on average in 93 milliseconds on CPU Intel Core i5-7200U@2.50 GHz using MATLAB R2018a. An optimal schedule for due dates (85) is

$$\mathbf{S}^* = [s_t^*]_{1\times 10} = [5 \quad 4 \quad 4 \quad 3 \quad 5 \quad 3 \quad 2 \quad 2 \quad 1 \quad 1] \tag{88}$$

and its total tardiness is

$$\vartheta^*(5, 2) = \sum_{n=1}^{5} \max\{0, \theta^*(n; 2) - d_n\}$$
$$= \max\{0, 10-10\} + \max\{0, 8-3\} + \max\{0, 6-4\} + \max\{0, 3-1\} + \max\{0, 5-5\} = 9,$$

whereas schedule (88) is found on average in 73 milliseconds on the same equipment. Surely, another optimal schedule

$$\mathbf{S}^* = [s_t^*]_{1\times 10} = [5 \quad 4 \quad 4 \quad 3 \quad 3 \quad 5 \quad 2 \quad 2 \quad 1 \quad 1], \tag{89}$$

corresponding to schedule (87) with

$$\vartheta^*(5, 2) = \sum_{n=1}^{5} \max\{0, \theta^*(n; 2) - d_n\}$$
$$= \max\{0, 10-10\} + \max\{0, 8-3\} + \max\{0, 5-4\} + \max\{0, 3-1\} + \max\{0, 6-5\} = 9,$$

is found in the same time span. Schedule (89) can be obtained from schedule (88) by aggregating job 3. It is seems that the difference between the computation times could be treated as a negligible one, although the descending job order is 26.5 % faster here. Indeed, it is so for a single or a few such scheduling problems, but after scheduling 1000 such instances the difference becomes very significant (20 seconds).

Another, more noticeably demonstrative, example is for scheduling 7 three-parted jobs whose due dates are

$$\mathbf{D} = [d_n]_{1\times 7} = [2 \quad 7 \quad 2 \quad 7 \quad 7 \quad 2 \quad 12] \tag{90}$$

for the ascending job order and

$$\mathbf{D} = [d_n]_{1\times 7} = [12 \quad 2 \quad 7 \quad 7 \quad 2 \quad 7 \quad 2] \tag{91}$$

for the descending job order. An optimal schedule for due dates (90) is

$$\mathbf{S}^* = [s_t^*]_{1\times 21} = [1 \quad 1 \quad 1 \quad 3 \quad 3 \quad 3 \quad 5 \quad 5 \quad 5 \quad 6 \quad 6 \quad 6 \quad 7 \quad 7 \quad 7 \quad 4 \quad 4 \quad 4 \quad 2 \quad 2 \quad 2] \tag{92}$$

whose total tardiness is

$$\vartheta^*(7,\,3) = \sum_{n=1}^{7} \max\{0,\, \theta^*(n;\,3) - d_n\}$$
$$= \max\{0,\,3-2\} + \max\{0,\,21-7\} + \max\{0,\,6-2\}$$
$$+ \max\{0,\,18-7\} + \max\{0,\,9-7\} + \max\{0,\,12-2\}$$
$$+ \max\{0,\,15-12\} = 45\,.$$

Schedule (92) is found on average in 69.51 seconds. On the other side, model (17)—(31) finds an optimal schedule for due dates (91), which is not obtained from schedule (92) by inversing the job number from n to $7-n+1$ for $n=\overline{1,\,7}$ (as it has been for the previous example with 5 jobs):

$$\mathbf{S}^* = [s_t^*]_{1\times 21} = [7\ \ 7\ \ 7\ \ 5\ \ 5\ \ 5\ \ 6\ \ 6\ \ 6\ \ 3\ \ 3\ \ 3\ \ 4\ \ 4\ \ 4\ \ 2\ \ 2\ \ 2\ \ 1\ \ 1\ \ 1] \tag{93}$$

whose total tardiness is

$$\vartheta^*(7,\,3) = \sum_{n=1}^{7} \max\{0,\, \theta^*(n;\,3) - d_n\}$$
$$= \max\{0,\,21-12\} + \max\{0,\,18-2\} + \max\{0,\,12-7\}$$
$$+ \max\{0,\,15-7\} + \max\{0,\,6-2\} + \max\{0,\,9-7\}$$
$$+ \max\{0,\,3-2\} = 45.$$

Meanwhile, schedule (93) is found on average in 36.52 seconds. This is almost twice faster than computing for due dates (90). Of course, schedule

$$\mathbf{S}^* = [s_t^*]_{1\times 21} = [7\ \ 7\ \ 7\ \ 5\ \ 5\ \ 5\ \ 3\ \ 3\ \ 3\ \ 2\ \ 2\ \ 2\ \ 1\ \ 1\ \ 1\ \ 4\ \ 4\ \ 4\ \ 6\ \ 6\ \ 6] \tag{94}$$

which can be obtained from schedule (92) by inversing the job number from n to $7-n+1$ for $n=\overline{1,\,7}$ is optimal also as its total tardiness is the same:

$$\vartheta^*(7,\,3) = \sum_{n=1}^{7} \max\{0,\, \theta^*(n;\,3) - d_n\}$$
$$= \max\{0,\,15-12\} + \max\{0,\,12-2\} + \max\{0,\,9-7\}$$
$$+ \max\{0,\,18-7\} + \max\{0,\,6-2\} + \max\{0,\,21-7\}$$
$$+ \max\{0,\,3-2\} = 45.$$

In fact, schedule (94) is found just as fast as schedule (93). In this example, the time saved on a single job scheduling problem owing to the efficient total tardiness exact minimization by the descending job order is about 32.99 seconds. Obviously, after scheduling a series of 1000 such instances the saved time exceeds 9 hours. Nevertheless, the computational study has also helped to reveal that the computation times may badly depend on the sufficiently great positive integer α taken for model (17)—(31) as sum (25). Thus, the described examples by significantly changing values α may come even with the reverse efficiency, where the ascending job order is faster.

Conclusions

It has been ascertained that the job order in tight-tardy progressive single machine scheduling with idling-free preemptions of equal-length jobs really influences the speed of computing the exact schedule whose total tardiness is minimal. Based on the pattern of generating instances of the job scheduling problem, in which for avoiding trivial schedules due dates are neither given in non-descending order, nor are given in non-ascending order, it has been revealed that scheduling a fewer jobs divided into a fewer job parts is executed on average faster by the descending job order. This is about scheduling 2 to 6 jobs divided into two to four or even five parts each, where the descending job order computation time can be shorter up to 10 % and more. However, there is no regularity in such an efficiency of the descending job order. Moreover, as the number of jobs increases along with increasing the number of their processing periods, the ascending job order becomes more efficient (i. e., faster than the descending job order) but its efficiency will be still irregular.

The research should be furthered by studying the case when the jobs have different processing periods. In this connection, the priority weights can be also considered for exactly minimizing total weighted tardiness. Additionally, the potential reverse efficiency accidentally revealed by significantly changing the infinity "substitute" is a matter for optimizing the Boolean linear programming model itself.

References

[1] P. Brucker, *Scheduling Algorithms.* Berlin, Heidelberg, Germany: Springer-Verlag, 2007, 371 p. doi: 10.1007/978-3-540-69516-5

[2] M.L. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Springer International Publishing, 2016, 670 p. doi: 10.1007/978-3-319-26580-3

[3] 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

[4] 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

[5] 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

[6] 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

[7] Z.J. Tian *et al.*, "An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness", *J. Scheduling*, vol. 9, iss. 4, pp. 343–364, 2006. doi: 10.1007/s10951-006-7039-6

[8] V.V. Romanuke, "The exact minimization of total weighted completion time in the preemptive scheduling problem by subsequent length-equal job importance growth", *Bulletin of V. Karazin Kharkiv National University. Series "Mathematical Modelling. Information Technology. Automated Control Systems"*, iss. 40, pp. 60–66, 2018. doi: 10.26565/2304-6201-2018-40-07

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

В.В. Романюк

ЕФЕКТИВНА ТОЧНА МІНІМІЗАЦІЯ ЗАГАЛЬНОГО ЗАПІЗНЮВАННЯ У ЩІЛЬНОМУ ПРОГРЕСУЮЧОМУ ОДНОМАШИННОМУ ПЛАНУВАННІ РІВНОЦІННИХ ЗАВДАНЬ ІЗ ПЕРЕМИКАННЯМИ БЕЗ ПРОСТОЮ

Проблематика. Розклад, що забезпечує строго мінімальне загальне 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 в модель у зворотному порядку.

Мета дослідження. Мета полягає у тому, щоб встановити, чи впливає на швидкість обчислення точного розв'язку порядок завдань у щільному прогресуючому одномашинному плануванні рівноцінних завдань із перемиканнями без простою. Для пошуку розкладів із мінімальним загальним запізнюванням використовується модель булевого лінійного програмування.

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 точної моделі, не розглядаються.

Результати дослідження. На основі тривимірного брусоподібного графіка відносної різниці між усередненими часами обчислень було показано, що існує можливість більш ефективного пошуку розкладів завдяки маніпулюванню порядком завдань. Наприклад, розклади 5-ти завдань, що складаються з двох періодів обробки, в середньому знаходяться на 14,67 % швидше для спадного порядку завдань. В іншому прикладі з 7-ми завдань, що складаються з трьох частин кожне, оптимальний розклад знаходиться в середньому за 69,51 секунди за висхідного порядку завдань, тоді як для спадного порядку завдань потрібно лише 36,52 секунди, що заощаджує 32,99 секунди.

Висновки. Планування меншої кількості завдань, розділених на меншу кількість частин, виконується в середньому швидше за спадного порядку завдань. Щойно кількість завдань збільшується разом зі збільшенням кількості періодів їх обробки, висхідний порядок завдань стає більш ефективним. Однак ефективність часу обчислення за обома порядками завдань має тенденцію до нерегулярності.

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

В.В. Романюк

ЭФФЕКТИВНАЯ ТОЧНАЯ МИНИМИЗАЦИЯ ОБЩЕГО ЗАПАЗДЫВАНИЯ В ПЛОТНОМ ПРОГРЕССИРУЮЩЕМ ОДНОМАШИННОМ ПЛАНИРОВАНИИ РАВНОЦЕННЫХ ЗАДАНИЙ С ПЕРЕКЛЮЧЕНИЯМИ БЕЗ ПРОСТОЯ

Проблематика. Расписание, обеспечивающее строго минимальное 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 исследования. Цель состоит в том, чтобы установить, влияет ли на скорость вычисления точного решения порядок заданий в плотном прогрессирующем одномашинном планировании равноценных заданий с переключениями без простоя. Для поиска расписаний с минимальным общим запаздыванием используется модель булевого линейного программирования.

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 точной модели, не рассматриваются.

Результаты исследования. На основе трехмерного брусоподобного графика относительной разности между усредненными временами вычислений было показано, что существует возможность более эффективного поиска расписаний путем манипулирования порядком заданий. Например, расписания 5-ти заданий, состоящих из двух периодов обработки, в среднем находятся на 14,67 % быстрее для нисходящего порядка заданий. В другом примере из 7-ми заданий, состоящих из трех частей каждое, оптимальное расписание находится в среднем за 69,51 секунды при восходящем порядке заданий, тогда как для нисходящего порядка заданий нужно лишь 36,52 секунды, что экономит 32,99 секунды.

Выводы. Планирование меньшего количества заданий, разделенных на меньшее количество частей, выполняется в среднем быстрее при нисходящем порядке заданий. Как только количество заданий увеличивается вместе с увеличением количества периодов их обработки, восходящий порядок заданий становится более эффективным. Однако эффективность времени вычисления при обоих порядках заданий имеет тенденцию к нерегулярности.

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

Рекомендована Радою
факультету прикладної математики
КПІ ім. Ігоря Сікорського

Надійшла до редакції
16 жовтня 2019 року

Прийнята до публікації
04 лютого 2020 року