J Sched (2016) 19:659–685
DOI 10.1007/s10951-016-0471-3

Necessary and sufficient optimality conditions for scheduling unit time jobs on identical parallel machines

Peter Brucker[1] · **Natalia V. Shakhlevich**[2]

Published online: 2 April 2016

Abstract In this paper we characterize optimal schedules for scheduling problems with parallel machines and unit processing times by providing necessary and sufficient conditions of optimality. We show that the optimality conditions for parallel machine scheduling are equivalent to detecting negative cycles in a specially defined graph. For a range of the objective functions, we give an insight into the underlying structure of the graph and specify the simplest types of cycles involved in the optimality conditions. Using our results we demonstrate that the optimality check can be performed by faster algorithms in comparison with existing approaches based on sufficient conditions.

Keywords Parallel machine scheduling · Unit processing times · Optimality conditions

1 Introduction

Finding optimal schedules is the primary goal of scheduling and therefore the main stream of research often deals with sufficient conditions of optimality that play the key role in the design of solution algorithms and in proving their correctness. The new trend is the study of the structural properties of optimal solutions. The practical aspects of this research direction are discussed in the recent paper by Lin and Wang (2007) which presents necessary and sufficient conditions of optimality for some classes of scheduling problems. As stated in Lin and Wang (2007), the optimality conditions may be useful for

- developing efficient algorithms that verify whether a given solution is optimal (such algorithms may be faster than the algorithms which construct optimal solutions);
- finding several (or all) optimal solutions;
- multi-objective hierarchical optimization (finding a characterization of all optimal solutions for the primary criterion and then optimizing the other one);
- developing new solution algorithms.

The most recent application of optimality conditions is related to solving inverse scheduling problems, which has become an important research area (see surveys Ahuja and Orlin 2001; Heuberger 2004). In an inverse scheduling problem, there is given a target solution, which may be non-optimal for initial values of problem parameters. The objective is to adjust problem parameters (e.g., to modify job weights) to make the target solution optimal. A typical approach for solving inverse problems is based on necessary and sufficient conditions of optimality which are used to produce a mathematical programming formulation of the inverse problem.

In this paper, we study a range of scheduling models with unit time operations. Given are m identical parallel machines and a set $J = \{1, 2, \ldots, n\}$ of n jobs with processing times $p_j = 1, j \in J$. Each job j has to be processed by one machine; it cannot start before some integer time $r_j \geq 0$ and it is desirable to complete it before a given due date d_j. A schedule $\mathbf{C} = (C_j)_{j=1}^n$ assigns to each job j a finishing time C_j (or starting time $S_j = C_j - 1$). $\mathbf{C}$ is feasible if

Peter Brucker tragically passed away on July 24, 2013. This paper is the outcome of our collaborative research and is dedicated to his memory.

✉ Natalia V. Shakhlevich
N.Shakhlevich@leeds.ac.uk

[1] Fachbereich Mathematik/Informatik, Universität Osnabrück, 49069 Osnabrück, Germany

[2] School of Computing, University of Leeds, Leeds LS2 9JT, UK

$r_j \leq C_j - 1$ for each job j and the number of jobs processed simultaneously at any time is at most m. We assume that all data are integer. The objective is to find a feasible schedule $\mathbf{C}$ which minimizes a given non-decreasing function $F(\mathbf{C}, \mathbf{w})$ depending on job completion times and job weights $\mathbf{w} = (w_j)_{j=1}^n$, which are assumed to be positive. The objective functions considered in this paper are

(a) total weighted completion time $\sum_{j=1}^n w_j C_j$;
(b) the weighted number of late jobs $\sum w_j U_j(C_j)$, where $U_j(C_j)$ is the unit penalty for completing job j after its due date d_j, $U_j(C_j) = 1$ if $C_j > d_j$ and $U_j(C_j) = 0$, otherwise;
(c) the weighted tardiness $\sum w_j T_j(C_j)$, where $T_j(C_j) = \max\{C_j - d_j, 0\}$ is the tardiness of job j.

The first function is strictly increasing while the last two functions are non-decreasing. In the last two cases we classify all jobs as *early* or *late*. Early jobs satisfy condition $C_j \leq d_j$ and they incur a zero cost.

Using standard three field notation, the scheduling problems we consider are denoted as $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$, where the first field represents identical parallel machines, the second field specifies job requirements and the third field is the objective function of type (a), (b) or (c). We replace P by 1 in the first field if there is a single machine ($m = 1$). We drop r_j from the second field if all jobs are available simultaneously ($r_j = 0$ for all $j \in J$) and drop $\mathbf{w}$ from the third field if all jobs have the same weight ($w_j = 1$ for all $j \in J$).

Our primary objective is to produce the necessary and sufficient conditions of optimality for versions (a)-(c) of problem $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$. Our study complements the earlier results for the following problems: $1||L_{\max}$, $1||\sum w_j C_j$, $F2||C_{\max}$, $1||\sum U_j$ (Lin and Wang 2007), and $P|r_j, p_j = 1|\sum w_j U_j$ (Dourado et al. 2009). Notice that in the first problem $L_{\max}$ denotes the maximum lateness objective function, $L_{\max} = \max_{j \in J}\{C_j - d_j\}$; in the third problem, $F2$ in the first field denotes the two-machine flow shop problem.

In this paper we study the problems with unit time jobs. We give a characterization of the block structure of optimal schedules (Sect. 2) and derive the necessary and sufficient conditions of optimality for the general problem $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$. We show in Sect. 2.3 that verifying these conditions results in detecting negative cycles in a specially defined network $\overline{N}(\mathbf{C})$ of the transportation problem. The number of nodes of that network is $\nu = \lceil \frac{n}{m} \rceil$ and the network can be constructed in $O(m\nu^2) = O\left(\frac{n^2}{m}\right)$ time.

In the subsequent sections, we explore the properties of network $\overline{N}(\mathbf{C})$ for the following problems:

- $P|r_j, p_j = 1|\sum w_j C_j$ (Sect. 3),
- $P|r_j, p_j = 1|\sum w_j U_j$ (Sect. 4) and its special case $P|p_j = 1|\sum w_j U_j$ (Sect. 5),
- $P|r_j, p_j = 1|\sum w_j T_j$ (Sect. 6) and its special cases $P|r_j, p_j = 1|\sum T_j$ (Sect. 7) and $P|p_j = 1|\sum T_j$ (Sect. 8).

Taking into account special features of the above problems, we derive stronger conditions than those known for a general transportation problem. For each problem we also give an insight into the underlying structure of the graph $\overline{N}(\mathbf{C})$ and specify the simplest types of cycles which are sufficient to consider in the optimality conditions: *two-node cycles* for problems $P|r_j, p_j = 1|\sum w_j C_j$ and $P|p_j = 1|\sum T_j$, *chain cycles* for problems $P|p_j = 1|\sum w_j U_j$ and $P|r_j, p_j = 1|\sum T_j$, *spiral cycles* for problem $P|r_j, p_j = 1|\sum w_j U_j$ and for its unweighted case $P|r_j, p_j = 1|\sum U_j$, see Sect. 2.3 for the definitions of special cycles. Conclusions and possible directions for future research are presented in Sect. 9.

Notice that conditions we formulate for problem $P|r_j, p_j = 1|\sum w_j U_j$ are different from the previously known, e.g., those presented in Dourado et al. (2009). In particular, we give a precise characterization of spiral cycles that should be examined in the underlying network. As the new conditions are more explicit and precise, they allow us to develop a fast algorithm for verifying optimality of a given solution (see Appendix 3).

Verifying the optimality of a given schedule $\mathbf{C}$ is one of the important application areas of our study. Instead of finding an optimal solution $\mathbf{C}^*$ by using a traditional scheduling algorithm and comparing $F(\mathbf{C}, \mathbf{w})$ and $F(\mathbf{C}^*, \mathbf{w})$, the necessary and sufficient conditions can be applied directly to $\mathbf{C}$ without constructing an optimal solution $\mathbf{C}^*$. We discuss the computational aspects of the optimality check for the problems under study at the end of the corresponding section, providing technical details in appendices. As we show, the new algorithms based on the necessary and sufficient conditions outperform those that find optimal schedules.

2 General structural properties of feasible schedules

In this section we discuss the general structural properties of feasible schedules. Depending on the objective function, it might be necessary to start jobs as early as possible if the function is strictly increasing, or the jobs can be postponed within some limits if the function is non-decreasing. For example, in any optimal schedule for problem $P|r_j, p_j = 1|\sum w_j C_j$ the maximum number of available jobs is allocated to each unit time interval $[t, t+1[$ starting with $t = \min_{j \in J}\{r_j\}$. In

the case of problems $P|r_j, p_j = 1|\sum w_j U_j$ and $P|r_j, p_j = 1|\sum w_j T_j$, an optimal schedule can often be modified by reshuffling and postponing early jobs without changing their "early" status; the resulting schedule still has the same value of the objective function.

In order to develop a uniform approach, we propose in Sect. 2.1 the concept of an *earliest start schedule*, which provides a convenient framework for formulating the necessary and sufficient conditions of optimality. Then in Sect. 2.2 we demonstrate how an arbitrary schedule with postponed jobs can be modified into an earliest start schedule.

2.1 Earliest start schedules

For non-decreasing objective functions, we are particularly interested in so-called *earliest start schedules* which contain the maximum number of jobs in each unit time interval $[t, t+1[$, $t \geq \min_{j\in J}\{r_j\}$. For such a schedule $\mathbf{C}$, there exists a partition of the job set J into disjoint sets $B_1, B_2, \ldots, B_\kappa$, called *block sets*, and a partition of the time line into corresponding *block intervals* $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\kappa$, where $\mathcal{T}_i$ consists of l_i consecutive time slots $[t, t+1[$,

$$t \in \mathcal{T}_i = \{t_i, t_i + 1, \ldots, t_i + l_i - 1\}$$

associated with the block set B_i. The pair $(B_i, \mathcal{T}_i)$ is a *block* of length l_i if

(i) its starting time t_i is equal to a minimum release date among all jobs from B_i, $t_i = \min\{r_j | j \in B_i\}$;
(ii) it includes all jobs j with $t_i \leq r_j \leq t_i + l_i - 1$ and none of such jobs is included in any subsequent block;
(iii) each time interval $[t, t+1[$, $t \in \mathcal{T}_i$, with the exception of the last one, contains exactly m jobs, while the last unit time interval contains at most m jobs.

The first condition implies that block $(B_i, \mathcal{T}_i)$ starts at the earliest possible start time t_i. Moreover, all jobs j scheduled in $[t_i, t_i + 1[$ have $r_j = t_i$. The second condition is the separating rule for two consecutive blocks. It ensures that the jobs that follow $(B_i, \mathcal{T}_i)$ cannot be moved to time intervals $\mathcal{T}_i$. The third condition characterizes the structure of a block; it states that each unit time slot should be saturated.

The blocks are numbered in the order they appear in the schedule: $t_1 < t_2 < \cdots < t_\kappa$. For a block set B_i, its length is $l_i = \left\lceil \frac{|B_i|}{m} \right\rceil$. An earliest start schedule can be constructed by Algorithm 'Calculate Blocks' presented in Appendix 1. At the beginning the algorithm considers the time slot $[t, t+1[$ defined by the minimal release time t. If the number z of jobs which can be scheduled in this time slot is at most m, then these jobs define the first block $(B_i, \mathcal{T}_i)$, $i = 1$, with $\mathcal{T}_1 = \{t\}$. Otherwise the block contains more than m jobs and it is obtained by considering the subsequent time intervals $[t, t+1[$ one by one, adding each time to the block set B_i those jobs which are released at time t and assigning the maximum number of jobs to $[t, t+1[$. Whenever the number of available jobs is m or less, the block $(B_i, \mathcal{T}_i)$ is finalized and the new block is started. In addition to the schedule, the algorithm also calculates a function $h : \mathcal{T} \to \{1, \ldots, m\}$, where $\mathcal{T}$ is the union of all sets $\mathcal{T}_i$ and $h(t)$ is the number of jobs scheduled in $[t, t+1[$ for each $t \in \mathcal{T}$. The algorithm can be implemented in $O(n \log n)$ time by sorting the jobs in non-decreasing order of their release dates in the beginning. Clearly schedule $\mathbf{C}$ constructed by Algorithm 'Calculate Blocks' is an *earliest start schedule* since the number of jobs $h(t)$ scheduled at time t is equal to m except for, probably, the last time interval of a block, which may contain less jobs.

The following example illustrates the algorithm.

Example 1 Consider an instance of the problem $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$ with $m = 2$ machines, $n = 14$ jobs and a non-decreasing objective function F of job completion times. For the release dates presented in the table

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
r_j	0	0	1	1	0	4	4	4	5	6	7	7	7	8

the Algorithm 'Calculate Blocks' finds the block sets $B_1 = \{1, 2, 3, 4, 5\}$, $B_2 = \{6, 7, 8, 9\}$, $B_3 = \{10\}$, $B_4 = \{11, 12, 13, 14\}$ and their starting times $t_1 = 0, t_2 = 4, t_3 = 6, t_4 = 7$. The corresponding sets $\mathcal{T}_j$ are $\mathcal{T}_1 = \{0, 1, 2\}$, $\mathcal{T}_2 = \{4, 5\}$, $\mathcal{T}_3 = \{6\}$, $\mathcal{T}_4 = \{7, 8\}$ and the number of jobs $h(t)$ scheduled in $[t, t+1[$ is

t	0	1	2	3	4	5	6	7	8
$h(t)$	2	2	1	–	2	2	1	2	2

A feasible earliest start schedule $\mathbf{C}$ with block sets B_1, B_2, B_3 and B_4 is shown in Fig. 1. The jobs which start exactly at their release dates are dashed; the remaining jobs start after their release dates.

If a schedule $\mathbf{C}$ is given by a list of unit time intervals and a list of jobs allocated to them, then conditions (i)–(iii) can

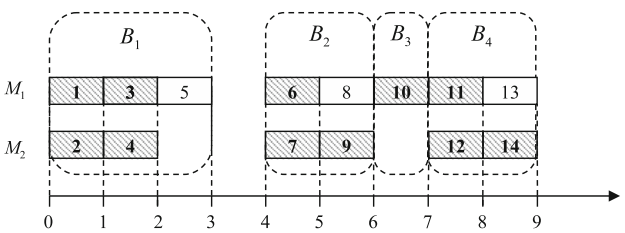


Fig. 1 The Gantt chart of schedule $\mathbf{C}$ found by Algorithm 'Calculate Blocks'

be verified for all time slots in $O(n)$ time by scanning the time slots of $\mathbf{C}$ three times.

In the first scan, the algorithm identifies time-values $\tau_1, \tau_2, \ldots, \tau_k$ with the property: all jobs that start at τ_u, $1 \leq u \leq k$, have their release times equal to τ_u. Clearly, the identified τ values satisfy property (i) of the block definition, and therefore they get a label of a 't-candidate.'

In the second scan, the algorithm removes from the list of 't-candidates' those values which do not satisfy property (ii): a 't-candidate' τ_u cannot define the starting time of a block if there is a job j starting after τ_u ($S_j > \tau_u$) which is related to an earlier block ($r_j < \tau_u$). To check condition (ii) efficiently, the 't-candidates' are scanned right to left updating ρ accordingly.

The third scan is performed to verify property (iii) for the remaining 't-candidates.'

The formal description of this approach is presented in Appendix 1 as Algorithm 'Earliest Start Schedule Verification.' Its time complexity is $O(n)$: there are no more than n unit time slots and they are scanned three times. The first two scans each consider n values of r_j; the third scan counts the number of jobs assigned to each time slot, which total number is n. Notice that as a by-product, the algorithm returns the t values, which define the decomposition of an earliest start schedule into the blocks.

2.2 Transforming an arbitrary schedule into an earliest start schedule

In this section we show that if a given schedule $\mathbf{C}$ for problem $P|r_j, p_j = 1|F$ does not belong to the class of earliest start schedules, it can be transformed into an earliest start schedule $\mathbf{C}'$ without increasing any of the completion times:

$$C'_j \leq C_j \text{ for all } j \in J. \tag{1}$$

This can be achieved as follows. Algorithm 'Calculate Blocks' is applied first in order to define the structure of the earliest start schedule, namely the time intervals $\mathcal{T}$ and the number of jobs allocated to them $h(t)$, $t \in \mathcal{T}$. Time intervals $[t, t+1[$, $t \in \mathcal{T}$, are considered one by one, starting with the earliest one. All jobs allocated in the original schedule to those time intervals are kept. If their number is less than $h(t)$, the required number of additional jobs is moved from later time intervals to $[t, t+1[$; the preference is given to the jobs with the smallest release dates. The formal description of such an algorithm (entitled as 'Left Shift($\mathbf{C}$)') and its analysis are presented in Appendix 1. We also discuss implementation details which result in the $O(n \log n)$ time complexity.

The following proposition establishes a link between a given optimal schedule and the earliest start optimal schedule.

Proposition 1 *A schedule* $\mathbf{C}$ *is optimal for problem* $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$ *with a non-decreasing objective function* F*, if and only if it can be transformed by Algorithm 'Left Shift* ($\mathbf{C}$)*' into an optimal earliest start schedule* $\mathbf{C}' = (C'_1, \ldots, C'_n)$ *without changing the value of the objective function.*

Proof Due to the properties of Algorithm 'Left Shift ($\mathbf{C}$),' schedules $\mathbf{C}$ and $\mathbf{C}'$ satisfy inequalities (1). This implies that for a non-decreasing objective function F,

$$F(\mathbf{C}', \mathbf{w}) \leq F(\mathbf{C}, \mathbf{w}). \tag{2}$$

If $\mathbf{C}$ is optimal, then condition (2) should hold as equality since a strict inequality contradicts optimality of $\mathbf{C}$. On the other hand, if $\mathbf{C}$ can be transformed into an optimal earliest start schedule $\mathbf{C}'$ without changing the objective function value, then clearly $\mathbf{C}$ must be optimal. □

Due to the described relationship between an arbitrary schedule $\mathbf{C}$ and an earliest start schedule $\mathbf{C}'$, in the subsequent sections we deal with earliest start schedules only. The optimality conditions which we formulate can be applied to each block separately, since no job from a block can be moved to a previous block.

Notice that if a given schedule does not belong to the class of earliest start schedules, then one can first apply Algorithm 'Left Shift ($\mathbf{C}$)' checking condition (2) for the resulting schedule $\mathbf{C}'$. If $F(\mathbf{C}', \mathbf{w}) < F(\mathbf{C}, \mathbf{w})$, then schedule $\mathbf{C}$ cannot be optimal; if $F(\mathbf{C}', \mathbf{w}) = F(\mathbf{C}, \mathbf{w})$ then one needs to verify the optimality of the earliest start schedule $\mathbf{C}'$ applying the necessary and sufficient conditions to it.

2.3 Earliest start schedule and associated compressed network

Consider problem $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$ with a separable objective function $F = \sum w_j f_j(C_j)$, where each function $f_j(C_j)$ is non-decreasing. For a given earliest start schedule $\mathbf{C}$ consisting of a single block, the total number of time intervals is

$$\nu = \left\lceil \frac{n}{m} \right\rceil,$$

so that $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$, and for each time $t \in \mathcal{T}$ there are exactly $h(t)$ jobs allocated to $[t, t+1[$:

$$h(t) = \begin{cases} m, & 0 \leq t \leq \nu - 2, \\ n - m(\nu - 1) \leq m, & t = \nu - 1. \end{cases}$$

It is well known that a scheduling problem with unit time jobs can be reformulated as a transportation problem. Any feasible schedule corresponds to a feasible flow in the associated network and the objective value of the schedule equals

the cost of delivering the flow. This implies that the necessary and sufficient conditions of optimality of a given schedule are equivalent to non-existence of a negative cycle in the corresponding residual network.

Formally, the transportation problem is defined by network $N(\mathbf{C}) = (V, A(\mathbf{C}), \eta)$,

$$V = \{1, \ldots, n\} \cup \{I_t | t \in \mathcal{T}\},$$
$$A(\mathbf{C}) = \{(j, I_t) \mid j \text{ is assigned or can be assigned to } I_t\}.$$

Here, the vertex set V consists of job nodes $\{1, \ldots, n\}$, each of which has a supply of 1, and interval nodes $\{I_t \mid t \in \mathcal{T}\}$ with demand $h(t)$. The arc set $A(\mathbf{C})$ consists of the arcs (j, I_t) defined for each feasible allocation of job j to interval $[t, t+1[$, $t \geq r_j$. The cost of an arc (j, I_t) represents the cost of allocating job j to time interval $[t, t+1[$:

$$\eta_{jt} = w_j \times f_j(t+1)$$

and its capacity is 1. Depending on the type of the objective function,

$$\eta_{jt} = \begin{cases} w_j \times (t+1), & \text{if } F = \sum w_j C_j, \\ w_j \times U_j(t+1), & \text{if } F = \sum w_j U_j, \\ w_j \times T_j(t+1), & \text{if } F = \sum w_j T_j. \end{cases}$$

For a given solution $\mathbf{C}$, a residual network $N^r(\mathbf{C}) = (V, A^r(\mathbf{C}), \xi)$ has the same vertex set V, while the arc set $A^r(\mathbf{C})$ and the costs ξ are defined as follows:

$$A^r(\mathbf{C}) = \{(I_t, j) \mid j \text{ is allocated in } \mathbf{C} \text{ to } [t, t+1[\}$$
$$\cup \left\{(j, I_t) \mid j \text{ is not allocated in } \mathbf{C} \text{ to } [t, t+1[, t \geq r_j\right\};$$
$$\xi_{tj} = -w_j \times f_j(t+1) \text{ for arcs } (I_t, j),$$
$$\xi_{jt} = w_j \times f_j(t+1) \text{ for arcs } (j, I_t).$$

Example 2 Consider problem $P|r_j, p_j = 1|\sum w_j T_j$ with $m = 2$ machines and $n = 5$ jobs with release dates, due dates and weights presented in the table:

j	1	2	3	4	5
r_j	0	0	1	1	0
d_j	2	1	2	3	1
w_j	1	1	1	1	1

Observe that the release dates are the same as those of the first 5 jobs of Example 1 processed as one block. The transportation network $N(\mathbf{C})$ is shown in Fig. 2a and the residual network corresponding to the schedule given by the first block of Fig. 1 is presented in Fig. 2b. The numbers at the arcs indicate the arc costs.

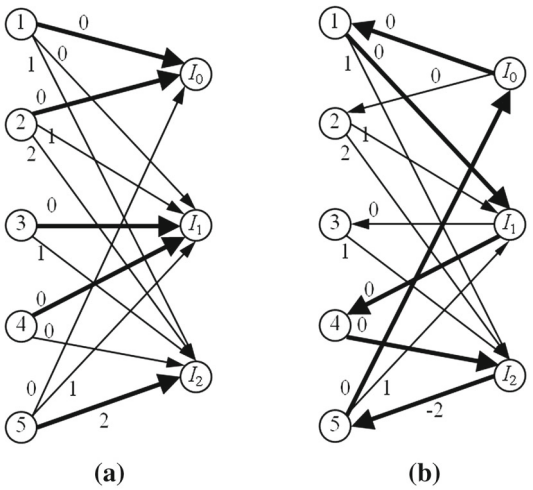


Fig. 2 **a** Network $N(\mathbf{C})$ for jobs $\{1, 2, 3, 4, 5\}$ of Example 2 and **b** residual network $N^r(\mathbf{C})$ for the schedule given by block B_1 of Fig. 1

Reformulating the well-known network flow theorem (see, e.g., Ahuja et al. 1993) we conclude that an earliest start schedule $\mathbf{C}$ is optimal for problem $P|r_j, p_j = 1|\sum w_j f_j(C_j)$ if and only if the corresponding residual network $N^r(\mathbf{C}) = (V, A^r(\mathbf{C}), \xi)$ constructed for each block of $\mathbf{C}$ contains no negative cycle.

A cycle in the residual network $N^r(\mathbf{C})$ is of the form $I_{t_1}, j_1, I_{t_2}, j_2, \ldots, I_{t_\kappa}, j_\kappa, I_{t_1}$. The arcs are alternating between an arc from an interval node I_t to a job node j and from a job node to an interval node.

It is convenient to compress the residual network $N^r(\mathbf{C})$ into a network $\widetilde{N}(\mathbf{C})$ by eliminating the job nodes; this results in multiple arcs connecting the interval nodes. We denote the nodes in $\widetilde{N}(\mathbf{C})$ by $t \in \{0, 1, \ldots, \nu - 1\}$. For any two nodes t and t' we have an arc from t to t' with length $\xi_{tj} + \xi_{jt'}$, where $\xi_{tj} + \xi_{jt'}$ is the change of the objective function $\sum w_j f_j(C_j)$ when job j is moved from $[t, t+1[$ to $[t', t'+1[$. Such an arc replaces the two alternating arcs $I_t \to j \to I_{t'}$ in $N^r(\mathbf{C})$. It is important to notice that the length $\ell_{(t,t')}$ of the arc from t to t' depends not only on t and t', but on job j allocated in $\mathbf{C}$ to $[t, t+1[$. Since there are $h(t)$ jobs allocated to each time interval, there are $h(t)$ arcs (t, t') for $t' > t$ since any job allocated to $[t, t+1[$ can be moved to a later time slot $[t', t'+1[$. However, for $t' < t$ there can be less than $h(t)$ arcs since not every job allocated to $[t, t+1[$ can be moved to an earlier time slot $[t', t'+1[$.

For the three objective functions we consider, the cost of the arc (t, t'), associated with job j allocated to $[t, t+1[$, is given by

$$\ell_{(t,t')} = \begin{cases} w_j \times (t' - t), & \text{if } F = \sum w_j C_j; \\ w_j \times \left(U_j\left(t'+1\right) - U_j(t+1)\right), & \text{if } F = \sum w_j U_j; \\ w_j \times \left(T_j\left(t'+1\right) - T_j(t+1)\right), & \text{if } F = \sum w_j T_j. \end{cases}$$

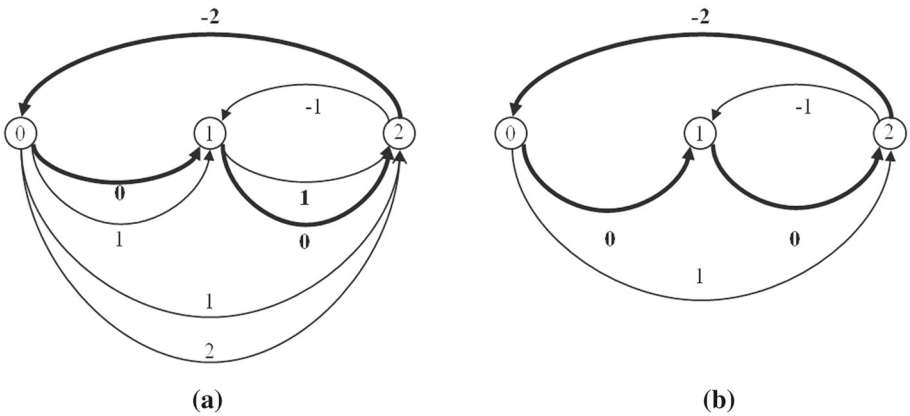


Fig. 3 Two compressed networks for Example 2: **a** $\widetilde{N}$ (**C**) and **b** $\overline{N}$ (**C**)

Example 3 Consider again the instance from Example 2. The corresponding residual network is presented in Fig. 2b and it contains a negative cycle $I_0,\ 1,\ I_1,\ 4,\ I_2,\ 5,\ I_0$ (its arcs are in bold) of length $-w_1 \max\{1-2,0\}+w_1 \max\{2-2,0\}-w_4 \max\{2-3,0\}+w_4 \max\{3-3,0\}-w_5 \max\{3-1,0\}+w_5 \max\{1-1,0\} = -2w_5 = -2$. This negative cycle characterizes that the series of the following moves leads to a schedule with a smaller value of $\sum w_j T_j$:

- moving job 1 from time interval $[0, 1[$ to time interval $[1, 2[$,
- moving job 4 from time interval $[1, 2[$ to time interval $[2, 3[$,
- moving job 5 from time interval $[2, 3[$ to time interval $[0, 1[$.

The compressed residual network is shown in Fig. 3a, and the corresponding negative cycle (marked by bold arcs) is $0, 1, 2, 0$.

We distinguish between right and left arcs in $\widetilde{N}(\mathbf{C})$. Arc (t, t') is a *left arc* if it is directed from t to a vertex corresponding to an earlier time slot $t', t' < t$; for a *right arc* $(t, t'), t' > t$.

Consider a pair of time-nodes t and t', corresponding to time intervals $[t, t+1[$ and $[t', t'+1[$. Since our aim is to detect negative cycles or to prove that none exists, we can compress network $\widetilde{N}(\mathbf{C})$ further by eliminating multiple arcs of larger costs. Instead of keeping multiple arcs (t, t') and multiple arcs (t', t), we can keep just one arc in each direction that has the smallest cost.

Denote the resulting network by $\overline{N}(\mathbf{C})$. For Example 2, the corresponding network $\overline{N}(\mathbf{C})$ is shown in Fig. 3b. Let J_t be the set of jobs assigned in $\mathbf{C}$ to time interval $[t, t+1[$. Then the cost (or the length) of the right arc $(t, t'), t < t'$, originating from t is defined as

$$\ell_{(t,t')} = \min_{j \in J_t} \left\{ w_j \times \left[f_j\left(t'+1\right) - f_j\left(t+1\right) \right] \right\}.$$

In particular, for $F = \sum w_j C_j$

$$\ell_{(t,t')} = \min_{j \in J_t} \left\{ w_j \right\} \times (t' - t), \tag{3}$$

for $F = \sum w_j U_j$

$$\ell_{(t,t')} = \begin{cases} \min\limits_{j \in J_t} \left\{ w_j \right\}, & \text{if} \quad t+1 \le d_j < t'+1 \text{ for all } j \in J_t, \\ 0, & \text{otherwise;} \end{cases} \tag{4}$$

for $F = \sum w_j T_j$

$$\ell_{(t,t')} = \min_{j \in J_t} \left\{ w_j \times \left[T_j\left(t'+1\right) - T_j\left(t+1\right) \right] \right\}.$$

Considering left arcs $(t, t'), t > t'$, originating from t, we need to take into account the release dates of the jobs from J_t since not all of them can be moved to an earlier time slot $[t', t'+1[$. Let $J_{(t,t')}$ denote a subset of jobs from J_t, which can be re-allocated to $[t', t'+1[$ without violating their release dates:

$$J_{(t,t')} = \left\{ j \in J_t | r_j \le t' \right\}.$$

Then the left arc $\left(t, t'\right)$ exists if $J_{(t,t')} \neq \emptyset$ and its length is defined as

$$\ell_{(t,t')} = - \max_{j \in J_{(t,t')}} \left\{ w_j \left[f_j\left(t+1\right) - f_j\left(t'+1\right) \right] \right\}.$$

In particular, for $F = \sum w_j C_j$

$$\ell_{(t,t')} = - \max_{j \in J_{(t,t')}} \left\{ w_j \right\} \times (t - t'), \tag{5}$$

for $F = \sum w_j U_j$

$$\ell_{(t,t')} = \begin{cases} 0, & \text{if } t'+1 > d_j \text{ or } t+1 \le d_j \\ & \text{for all } j \in J_{(t,t')}, \\ -\max\limits_{\substack{j \in J_{(t,t')}, \\ t'+1 \le d_j < t+1}} \{w_j\}, & \text{otherwise;} \end{cases}$$

for $F = \sum w_j T_j$

$$\ell_{(t,t')} = -\max_{j \in J_{(t,t')}} \left\{ w_j \left[T_j(t+1) - T_j\left(t'+1\right) \right] \right\}.$$

It is easy to make sure that if residual network $N^r(\mathbf{C})$ and the corresponding compressed network $\overline{N}(\mathbf{C})$ contain a negative cycle with repeated node t, then the cycle can be decomposed into two cycles and at least one of them is negative. Clearly, it is sufficient to consider the cycles without repeated time-values. Thus the necessary and sufficient optimality conditions can be formulated as follows.

Theorem 1 *A schedule* $\mathbf{C}$ *for a given instance of* $P|r_j, p_j = 1|\sum w_j f_j(C_j)$ *is optimal if and only if*

1. *transforming* $\mathbf{C}$ *into an earliest start schedule does not change the objective value;*
2. *for each block of the resulting earliest start schedule, the corresponding compressed network* $\overline{N}(\mathbf{C})$ *contains no negative cycle.*

We estimate the size of the compressed network $\overline{N}(\mathbf{C})$ and the time complexity of constructing it assuming that a given earliest start schedule $\mathbf{C}$ has n jobs assigned to m machines and it consists of a single block. The number of nodes of $\overline{N}(\mathbf{C})$ is $\nu = \lceil \frac{n}{m} \rceil$. Each pair of nodes t and t' are connected by at most two arcs (t, t') and (t', t), so that the total number of arcs is $O\left(\nu^2\right)$. For each right arc (t, t'), its cost can be found in $O(m)$ time since $|J_t| \le m$, and for each left arc (t, t'), its cost can also be found in $O(m)$ time since $|J_{(t,t')}| \le m$. Thus the overall time complexity of constructing $\overline{N}(\mathbf{C})$ is $O\left(m\nu^2\right) = O\left(\frac{n^2}{m}\right)$.

In the subsequent sections we show that, depending on the type of the problem, Condition 2 of Theorem 1 can be reformulated so that instead of checking the non-negative condition for all possible cycles only special types of cycles can be considered, namely two-node cycles, chain cycles and spiral cycles. Each of these cycles has exactly one arc of negative length denoted by (t, τ); it originates in the right-most node t of the cycle and contains no more than one positive arc; all other arcs have zero lengths. If the cycle contains a positive arc, that arc has end-node t.

A *two-node cycle* consists of two arcs (t, τ) and (τ, t), see Fig. 4.

A *chain cycle* consists of an arc in one direction and a chain of arcs in the opposite direction. A *right-chain cycle* starts with the negative left arc (t, τ) and then proceeds with a chain R of right arcs from τ to t, see Fig. 5a. A *left-chain cycle* starts with the negative left arc (t, τ), proceeds then with a chain L of left arcs to the left-most node of the cycle and returns to the origin t via one right arc, see Fig. 5b.

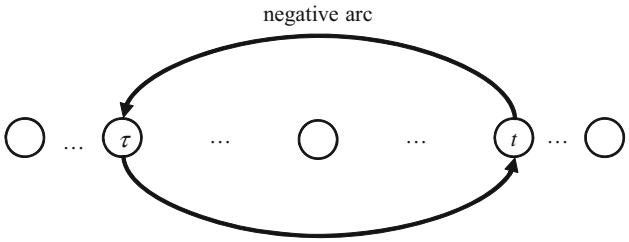


Fig. 4 Two-node cycle

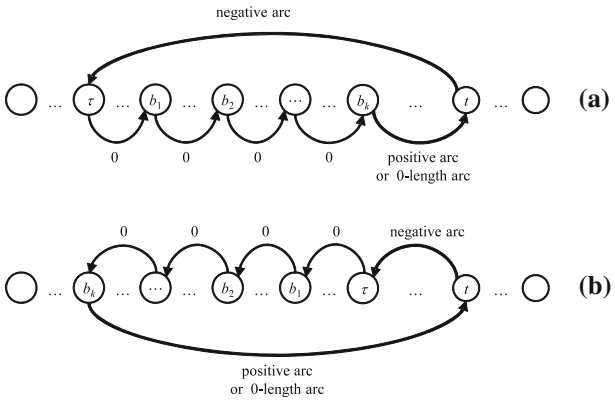


Fig. 5 Chain cycles: **a** right-chain cycle and **b** left-chain cycle

A *spiral cycle* is defined as the one satisfying the following five properties.

Property 1: The cycle contains exactly one negative arc (t, τ), all other left arcs have zero length.

Property 2: No left arc, except for possibly (t, τ), spans over a node which appears in the remaining part of the cycle after that arc.

Property 3: No right arc spans over a node which appears in the remaining part of the cycle after that arc.

Property 4: The right-most node of the cycle is t.

Property 5: All right arcs have zero length except for the last right arc terminating in t which may be of positive length or of zero length.

An example of a spiral cycle is shown in Fig. 6. In general, a spiral cycle contains alternating left chains $L_1, L_2, \ldots, L_s$ and right chains $R_1, R_2, \ldots, R_s$: the negative left arc (t, τ) is followed by $L_1, R_1, L_2, R_2, \ldots, L_s, R_s$ terminating in t, see Fig. 6. L_1 is a left chain originating in τ; the first arc of R_1 spans over all nodes of L_1; the first arc of L_2 spans over all nodes of $L_1 \cup R_1$. For each $k = 2, \ldots, s$, the first arc of L_k spans over all nodes of $L_1, R_1, L_2, R_2, \ldots, L_{k-1}, R_{k-1}$

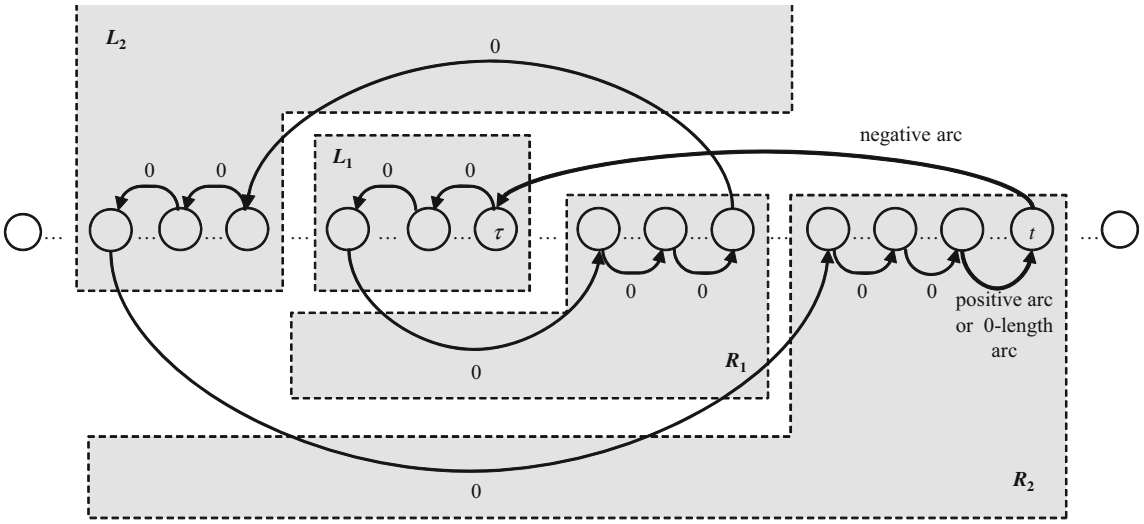


Fig. 6 Spiral cycle

Table 1 The simplest cycles which are sufficient to consider in optimality conditions

Problem	Simplest negative cycles	Reference
$P\|r_j, p_j = 1\|\sum w_j C_j$	Two-node cycles	Theorem 2
$P\|r_j, p_j = 1\|\sum w_j U_j$	Spiral cycles (even if $w_j = 1$)	Theorem 4, Example 5
$P\|p_j = 1\|\sum w_j U_j$	Right-chain cycles (even if $w_j = 1$)	Theorem 5, Example 6
$P\|r_j, p_j = 1\|\sum w_j T_j$	Cycles with more than one negative arc (even if $r_j = 0$)	Example 8
$P\|r_j, p_j = 1\|\sum T_j$	Left-chain cycles	Theorem 7
$P\|p_j = 1\|\sum T_j$	Two-node cycles	Theorem 8

and the first arc of R_k spans over all nodes of L_1, R_1, L_2, R_2, …, L_{k-1}, R_{k-1}, L_k. The last chain R_s consists of right arcs terminating in the origin t.

Notice that a two-node cycle is a special case of a chain cycle, which in its turn is a special case of a spiral cycle.

The main outcomes of our study are summarized in Table 1, where we list the types of cycles that should be examined in a compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule. Notice that for problem $P|r_j, p_j = 1|\sum w_j T_j$ and its special case $P|p_j = 1|\sum w_j T_j$ there exist instances for which even the simplest negative cycles have several negative arcs so that they do not fall in any of the above three categories of cycles. We do not give a characterization of negative cycles for these two problems since the usage of such a result is likely to be very limited. For example, using those conditions in the optimality check would incur cycles with one negative arc, cycles with pairs of negative arcs and even more complex cycles with various combinations of negative arcs.

For all other problems and their special cases, we first prove the result establishing the type of a simplest negative cycle. Then we show that the result cannot be improved by providing an example that there exists an instance of the problem with only one negative cycle and it is of the type established in the corresponding theorem. Such instances are not needed for two-node cycles indeed.

In what follows we consider the three objective functions $\sum w_j C_j$, $\sum w_j U_j$ and $\sum T_j$ and prove the relevant statement about the type of the cycle for each problem. We then explore how the identified types of cycles can be used in order to speed up the optimality check.

3 Problem $P|r_j, p_j = 1|\sum w_j C_j$

In this section we consider the problem $P|r_j, p_j = 1|\sum w_j C_j$ with the weighted completion time objective. We show that the optimality conditions for this problem can be simplified by limiting consideration to two-node cycles only. Moreover, the reduced network $\overline{N}(\mathbf{C})$ can be compressed further by eliminating transitive arcs, so that the resulting network $\overline{N}_{\text{TransRem}}(\mathbf{C})$ contains no more than $2(v-1)$ arcs; its left arcs form an out-tree and the right arcs form an in-tree. As a result, the optimality check problem can be solved in $O(n)$ time, an improvement in comparison with the $O(n \log n)$ time algorithm for finding an optimal schedule.

Theorem 2 *A compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule for a given instance of problem $P|r_j, p_j = 1|\sum w_j C_j$, contains a negative cycle if and only if it contains a negative two-node cycle.*

Proof Clearly, the formulated condition is sufficient. In order to prove that it is also necessary, assume that there exists an instance of problem $P|r_j, p_j = 1|\sum w_j C_j$ such that all two-node cycles are non-negative while there exists a negative cycle with three or more nodes. Let this cycle additionally have the smallest number of nodes. We show that there exists a negative cycle with a smaller number of nodes.

Denote by t_1 the left-most node of the cycle. The cycle starts with a positive arc (t_1, t_2), proceeds with a series of arcs, which we denote by α, leading to node t_3 and terminates at the origin t_1 with a negative arc (t_3, t_1). The length of the positive arc (t_1, t_2) is $w_{j_1}(t_2 - t_1)$, where job j_1 is selected in accordance with (3) so that

$$w_{j_1} = \min_{j \in J_{t_1}} \{w_j\};$$

the length of the negative arc (t_3, t_1) is $-w_{j_3}(t_3 - t_1)$, where job j_3 is selected in accordance with (5) so that

$$w_{j_3} = \max_{j \in J_{(t_3,t_1)}} \{w_j\}. \tag{6}$$

Thus the length of the original cycle is

$$\ell = w_{j_1}(t_2 - t_1) + \ell_\alpha - w_{j_3}(t_3 - t_1), \tag{7}$$

where ℓ_α is the length of fragment α.

First we derive an auxiliary inequality for w_{j_1} and w_{j_3}. Due to the assumption, the cycle consisting of two arcs (t_3, t_1) and (t_1, t_3) is non-negative. Since its length is $(-w_{j_3} + w_{j_1})(t_3 - t_1)$, we conclude that

$$-w_{j_3} + w_{j_1} \geq 0. \tag{8}$$

Consider the following two cases, depending on the location of t_2 and t_3. If $t_2 < t_3$, cycle $((t_1, t_2), \alpha, (t_3, t_1))$ is of the form shown in Fig. 7a. Since there exists arc (t_3, t_1) in $\overline{N}(\mathbf{C})$, the set of jobs $J_{(t_3,t_1)}$, which can be re-allocated from $[t_3, t_3 + 1[$ to $[t_1, t_1 + 1[$ is non-empty, and therefore the set $J_{(t_3,t_2)}$ is also non-empty for $t_2 > t_1$. Hence we can introduce the arc (t_3, t_2) and consider a cycle with less nodes, namely $(\alpha, (t_3, t_2))$, see Fig. 7b. The length of (t_3, t_2) is associated with some job j_3', $w_{j_3'} = \max_{j \in J_{(t_3,t_2)}} \{w_j\}$, and due to $J_{(t_3,t_2)} \supseteq J_{(t_3,t_1)}$,

$$w_{j_3'} \geq w_{j_3}. \tag{9}$$

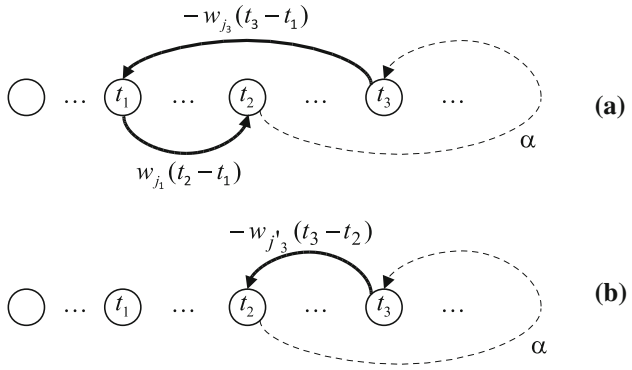


Fig. 7 Replacing negative cycle $((t_1, t_2), \alpha, (t_3, t_1))$ by $(\alpha, (t_3, t_2))$ if $t_1 < t_2 < t_3$

Comparing the length of the original cycle ℓ given by (7) with the length of the new cycle $\ell' = \ell_\alpha - w_{j_3'}(t_3 - t_2)$ we conclude that if $\ell < 0$, the new cycle is also negative:

$$\begin{aligned}\ell' - \ell &= \left[\ell_\alpha - w_{j_3'}(t_3 - t_2)\right] \\ &\quad - \left[w_{j_1}(t_2 - t_1) + \ell_\alpha - w_{j_3}(t_3 - t_1)\right] \\ &\leq -w_{j_3}(t_3 - t_2) - \left[w_{j_1}(t_2 - t_1) - w_{j_3}(t_3 - t_1)\right] \\ &= \left(w_{j_3} - w_{j_1}\right)(t_2 - t_1) \leq 0.\end{aligned}$$

In the above formula, the first inequality is due to (9), while the second inequality follows from (8). Thus we have detected a negative cycle with a less number of nodes which contradicts the assumption that the initial negative cycle has the smallest number of nodes.

If $t_2 > t_3$, as shown in Fig. 8a, introduce the positive arc (t_3, t_2) and consider a cycle with less nodes, namely $((t_3, t_2), \alpha)$, see Fig. 8b. The length of (t_3, t_2) is associated with some job j_3'', $w_{j_3''} = \min_{j \in J_{t_3}} \{w_j\}$, see (3). Comparing the latter formula with (6) we conclude that

$$w_{j_3''} = \min_{j \in J_{t_3}} \{w_j\} \leq \min_{j \in J_{(t_3,t_1)}} \{w_j\} \leq \max_{j \in J_{(t_3,t_1)}} \{w_j\} = w_{j_3} \tag{10}$$

(notice that $J_{t_3} \supseteq J_{(t_3,t_1)}$ and the last equality corresponds to (6)).

Denoting by ℓ'' the length of the cycle shown in Fig. 8b, we obtain

$$\begin{aligned}\ell'' - \ell &= \left[w_{j_3''}(t_2 - t_3) + \ell_\alpha\right] \\ &\quad - \left[w_{j_1}(t_2 - t_1) + \ell_\alpha - w_{j_3}(t_3 - t_1)\right] \\ &\leq w_{j_3}(t_2 - t_3) - \left[w_{j_1}(t_2 - t_1) - w_{j_3}(t_3 - t_1)\right] \\ &= \left(w_{j_3} - w_{j_1}\right)(t_2 - t_1) \leq 0,\end{aligned}$$

so that $\ell'' < 0$ for $\ell < 0$. In the above formula, the first inequality is due to (10), while the second inequality follows

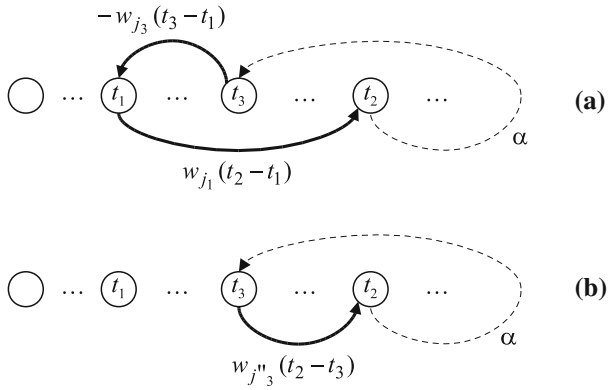


Fig. 8 Replacing negative cycle $((t_1,t_2),\alpha,(t_3,t_1))$ by $((t_3,t_2),\alpha)$ if $t_1 < t_3 < t_2$

from (8). Again we have detected a negative cycle with a less number of nodes which contradicts the assumption that the initial negative cycle has the smallest number of nodes. □

Notice that due to the definition of $\overline{N}(\mathbf{C})$, a right arc exists for any pair of nodes, while left arc (t,τ) exists only if $J_{(t,\tau)} \neq \emptyset$. Therefore in order to enumerate all two-node cycles one can consider left arcs one by one, complementing each left arc with its right arc counterpart. Moreover, we prove that transitive left arcs in $\overline{N}(\mathbf{C})$ are not needed.

Theorem 3 *Consider network $\overline{N}_{\text{TransRem}}(\mathbf{C})$ obtained from $\overline{N}(\mathbf{C})$ by removing transitive left arcs and removing all right arcs which do not have a left counterpart. Denote the set of left arcs in $\overline{N}_{\text{TransRem}}(\mathbf{C})$ by $A_{\text{left}}(\mathbf{C})$. Then $\overline{N}(\mathbf{C})$ does not contain a negative cycle if and only if inequality*

$$\min_{j\in J_\tau}\{w_j\} \geq \max_{j\in J_{(t,\tau)}}\{w_j\}. \tag{11}$$

is satisfied for each $(t,\tau)\in A_{\text{left}}(\mathbf{C})$, $\tau < t$.

Proof **Necessity** Clearly, relation (11) formulated for a left arc (t,τ) is an equivalent representation of the condition that the cycle defined by nodes t and τ is non–negative, see formulae (3) and (5).

Sufficiency To prove that conditions (11) defined for $(t,\tau)\in A_{\text{left}}(\mathbf{C})$, are sufficient, consider $t_1 < t_2 < t_3$ assuming that the left arcs $(t_2,t_1),(t_3,t_2)$ exist together with the transitive left arc (t_3,t_1). The associated positive arcs are $(t_1,t_2),(t_2,t_3)$ and (t_1,t_3). We show that if (11) holds for the pair of nodes t_1,t_2 and for the pair t_2,t_3, then (11) also holds for the pair of nodes t_1,t_3.

Indeed, conditions (11) for t_1,t_2 and for t_2,t_3 are of the form:

$$\min_{j\in J_{t_1}}\{w_j\} \geq \max_{j\in J_{(t_2,t_1)}}\{w_j\},$$
$$\min_{j\in J_{t_2}}\{w_j\} \geq \max_{j\in J_{(t_3,t_2)}}\{w_j\}.$$

The right-hand side of the first inequality is greater than or equal to the left-hand side of the second inequality since $J_{(t_2,t_1)}\subseteq J_{t_2}$:

$$\max_{j\in J_{(t_2,t_1)}}\{w_j\} \geq \min_{j\in J_{(t_2,t_1)}}\{w_j\} \geq \min_{j\in J_{t_2}}\{w_j\}.$$

The right-hand side of the second inequality can be bounded as

$$\max_{j\in J_{(t_3,t_2)}}\{w_j\} \geq \max_{j\in J_{(t_3,t_1)}}\{w_j\}$$

since $J_{(t_3,t_2)}\supseteq J_{(t_3,t_1)}$.

We conclude that

$$\min_{j\in J_{t_1}}\{w_j\} \geq \max_{j\in J_{(t_3,t_1)}}\{w_j\},$$

which implies that condition (11) also holds for the pair of nodes t_1,t_3. □

Proposition 2 *The set of left arcs $A_{\text{left}}(\mathbf{C})$ in $\overline{N}_{\text{TransRem}}(\mathbf{C})$ is an out-tree.*

Proof Suppose the proposition does not hold, i.e., there exists in $A_{\text{left}}(\mathbf{C})$ a pair of left arcs (t_2,t_1) and (t_3,t_1) with the same end-node, $t_1 < t_2 < t_3$. This implies that $J_{(t_2,t_1)}\neq\emptyset$ and $J_{(t_3,t_1)}\neq\emptyset$. It follows from the latter condition that $J_{(t_3,t_2)}\neq\emptyset$ since the jobs that can be re-allocated from $[t_3,t_3+1[$ to $[t_1,t_1+1[$ can also be re-allocated to $[t_2,t_2+1[$. Then there exists arc (t_3,t_2) in $\overline{N}(\mathbf{C})$ and therefore the set of arcs $A_{\text{left}}(\mathbf{C})$ in network $\overline{N}_{\text{TransRem}}(\mathbf{C})$ (without transitive arcs) cannot contain (t_3,t_1), a contradiction. □

Example 4 The following table describes an instance of problem $P|r_j,\ p_j=1|\sum w_jC_j$ with $n=18$ jobs and $m=2$ machines:

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
r_j	0	0	0	0	2	2	3	1	3	2	5	5	5	4	7	5	6	7
C_j	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9

A feasible schedule $\mathbf{C}$ for this problem is shown in Fig. 9. The jobs which start exactly at their release dates are dashed; the remaining jobs start after their release dates.

The left arcs $A_{\text{left}}(\mathbf{C})$ of graph $\overline{N}_{\text{TransRem}}(\mathbf{C})$ with transitive arcs removed are shown in Fig. 10. The set $A_{\text{left}}(\mathbf{C})$ consists of $(8,7),(7,6),(6,5),(6,4),(4,3),(3,2),(3,1),(1,0)$, which form an out-tree.

The proof of Proposition 2 justifies the following $O(n)$-time algorithm for constructing left arcs $A_{\text{left}}(\mathbf{C})$ of network

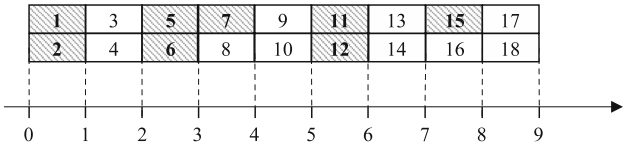


Fig. 9 Gantt chart of a feasible schedule **C** of Example 4

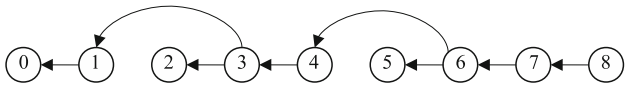


Fig. 10 Left arcs $A_{\text{left}}(\mathbf{C})$ of graph $\overline{N}_{\text{TransRem}}(\mathbf{C})$ without transitive arcs for a feasible schedule **C** of Example 4

$\overline{N}_{\text{TransRem}}(\mathbf{C})$ without transitive arcs. It first connects consecutive nodes into left chains and then defines for the right-most node of each chain an arc ending in that node with the origin in the subsequent chain, giving preference to the left-most possible origin. Having constructed the set $A_{\text{left}}(\mathbf{C})$ of left arcs, the corresponding right arcs can be produced also in $O(n)$ time. Since $|A_{\text{left}}(\mathbf{C})| = \nu - 1$, there are $\nu - 1$ two-node cycles, which can be enumerated in $O(\nu)$ time, so that the overall time complexity of the optimality check is $O(n)$. We provide the details of this approach in Appendix 2.

4 Problem $P|r_j, p_j = 1|\sum w_j U_j$

This section explores the most general, spiral cycles.

Theorem 4 *A compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule for problem $P|r_j, p_j = 1|\sum w_j U_j$, contains a negative cycle if and only if it contains a negative spiral cycle.*

Proof Consider an arbitrary negative cycle that starts with a negative arc (t, τ). We show that then there exists a negative cycle such that the five properties formulated in the definition of a spiral cycle are satisfied.

In order to prove Property 1 (the cycle contains exactly one negative arc (t, τ)) we demonstrate that a negative cycle with the smallest number of nodes cannot contain more than one negative arc. Suppose a negative cycle has several negative arcs $(t_1, \tau_1), (t_2, \tau_2), \ldots, (t_y, \tau_y)$, $y \geq 2$. Without loss of generality we assume that arc (t_1, τ_1) has the right-most origin t_1 among all negative arcs, i.e., $t_1 > \max\{t_2, \ldots, t_y\}$. Notice that no nodes are repeated in the cycle. Denoting the path from τ_1 to t_y by α and the path from τ_y to t_1 by β, the cycle can be represented in the form $((t_1, \tau_1), \alpha, (t_y, \tau_y), \beta)$, where fragment α can be empty. The three possible types of that cycle are illustrated in Fig. 11a–c depending on the relationship between τ_1, t_y and τ_y. Let the lengths of arcs (t_1, τ_1) and (t_y, τ_y) be $-w_{j_1}$ and $-w_{j_y}$, where j_1 and j_y are late jobs allocated to $[t_1, t_1 + 1[$ and $[t_y, t_y + 1[$, respectively.

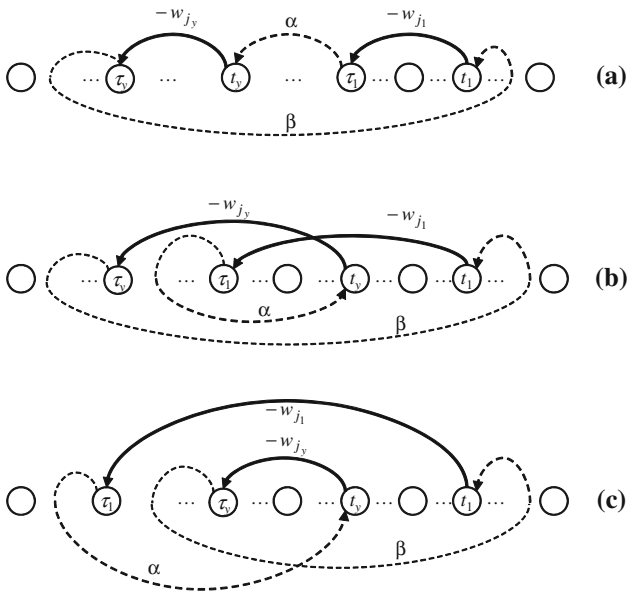


Fig. 11 Negative cycle $((t_1, \tau_1), \alpha, (t_y, \tau_y), \beta)$

Since the cycle passes through t_y before it returns to t_1 and $t_y < t_1$, there should be an arc (e, f) belonging to fragment β that straddles node t_y, so that

$$e < t_y < f,$$

and we can represent fragment β as $(\beta', (e, f), \beta'')$, see Fig. 12. Notice that the case $e = \tau_y$ implies $\beta' = \emptyset$ and the case $f = t_1$ implies $\beta'' = \emptyset$. Let the job corresponding to arc (e, f) be q so that the length of the arc is

$$\ell_{(e,f)} = \begin{cases} w_q, & \text{if} \quad e < d_q \leq f, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Since fragment β does not contain negative arcs, all its three components are non-negative:

$$\begin{aligned} \ell_{\beta'} &\geq 0, \\ \ell_{(e,f)} &\geq 0, \\ \ell_{\beta''} &\geq 0. \end{aligned}$$

Here $\ell_{\beta'}$, $\ell_{(e,f)}$ and $\ell_{\beta''}$ denote the lengths of β', (e, f), β'', respectively.

Consider instead of the original cycle two new cycles: $Cycle_I = \left((t_y, \tau_y), \beta', (e, t_y)\right)$ and $Cycle_{II} = ((t_1, \tau_1), \alpha, (t_y, f), \beta'')$. Notice that

$$\ell_{(e,t_y)} \leq \ell_{(e,f)}$$

since $t_y < f$. Moreover

$$\ell_{(t_y,f)} = 0$$

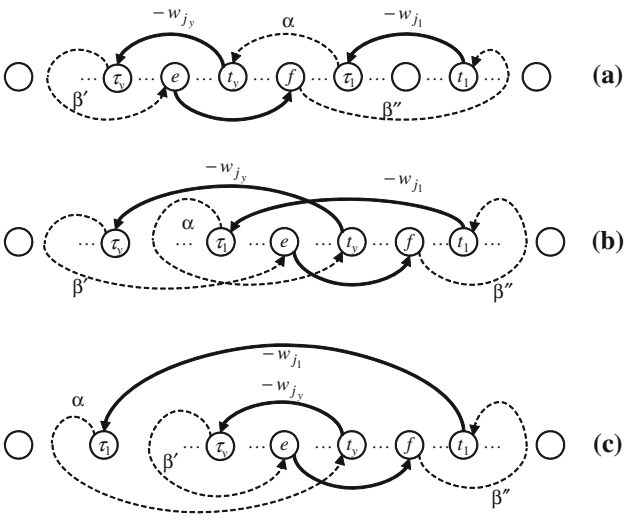


Fig. 12 Negative cycle with fragment β decomposed into $(\beta', (e, f), \beta'')$

since J_{t_y} contains a late job (it defines the cost of the negative left arc (t_y, τ_y)). Thus we have

$$\ell_{(e,t_y)} + \ell_{(t_y,f)} \leq \ell_{(e,f)}.$$

Denoting the length of the original cycle by ℓ_O and the lengths of the two new cycles by ℓ_I and ℓ_{II} we obtain:

$$\ell_I + \ell_{II} \leq \ell_O < 0. \tag{13}$$

Thus at least one of the new cycles is negative. This contradicts the assumption that the initial negative cycle has the smallest number of nodes.

We turn now to Property 2: no left arc, except for possibly (t, τ), spans over a node which appears in the remaining part of the cycle after that arc. Suppose for a negative cycle satisfying Property 1 there exists a 0-length left arc (f, g) which spans over node h that appears on the path from g to t, so that $g < h < f$, see Fig. 13. If there are several nodes of type h we select the right-most one. In that figure, solid lines represent arcs and dotted lines represent the fragments of the cycle which may consists of several arcs. Introduce new left arc (f, h), which length is zero since there is only one negative left arc (t, τ). Replacing the fragment of the cycle from f to h by the arc (f, h), we remove 0-length left arcs and non-negative right arcs. Hence the resulting cycle is negative, but with less left arcs spanning over other nodes on the path to t. Repeating this transformation we eventually obtain a cycle satisfying Property 2.

The proof of Property 3 (no right arc spans over a node which appears in the remaining part of the cycle after that arc) is similar to the proof of Property 2: consider a right arc (f, g) which spans over node h that appears on the path from

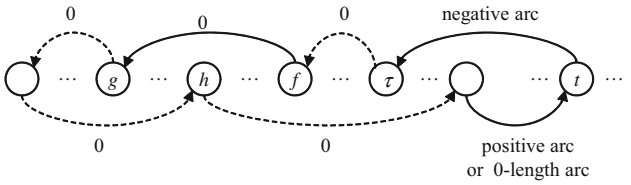


Fig. 13 Negative cycle with left arc (f, g) spanning over node h on the path from g to t

g to t, so that $f < h < g$, and replace the fragment of the cycle from f to h by the arc (f, h) with $\ell_{(f,h)} \leq \ell_{(f,g)}$.

The same idea can be used to prove Property 4 (t is the right-most node of the cycle) using t instead of h in the above arguments.

Finally, consider Property 5: all right arcs have zero length except for the last right arc terminating in t which may be of positive length or of zero length. Let there exists a negative cycle satisfying Properties 1–4, but not satisfying Property 5. Let the origin of the negative arc be t and let the first positive arc be (e, f), where $f < t$ due to Property 4. Again we introduce a new arc (e, t). For the positive arc (e, f) all jobs from J_e are early and they become late if re-allocated to $[f, f + 1[$, $\ell_{(e,f)} = \min_{j \in J_e}\{w_j\}$, see (4). Then the same holds for (e, t), which implies

$$\ell_{(e,t)} = \min_{j \in J_e}\{w_j\} = \ell_{(e,f)}.$$

Thus replacing the fragment from e to t by arc (e, t) results in a negative cycle with less nodes and no more than one positive arc (e, t). □

Clearly, spiral cycles have a more complex structure in comparison with two-node cycles and chain cycles. We demonstrate by means of Example 5 below that the result of Theorem 4 cannot be strengthened for problem $P|r_j, p_j = 1|\sum w_j U_j$, and spiral cycles cannot be replaced by simpler counterparts. Since the instance below deals with the case of unit weights, $w_j = 1$ for all $j \in J$, the result of Theorem 4 cannot be strengthened for problem $P|r_j, p_j = 1|\sum U_j$.

Example 5 Consider an instance of the problem $P|r_j, p_j = 1|\sum U_j$ for which a negative spiral cycle exists but there are no negative cycles of simpler types. In that example, there is one machine and $n = 8$ jobs with job characteristics given by the table:

j	1	2	3	4	5	6	7	8
r_j	0	1	2	2	1	0	6	3
d_j	7	6	5	4	5	6	8	4
C_j	1	2	3	4	5	6	7	8

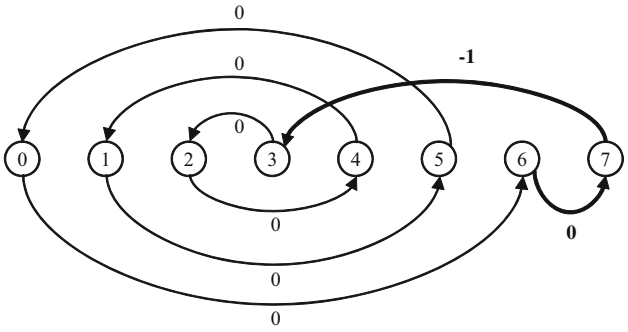


Fig. 14 A unique negative cycle for Example 5 belonging to the spiral category

Figure 14 represents a spiral negative cycle for this instance. Notice that (7, 3) is the unique negative arc in the graph and its length is -1. In order to construct a negative cycle with that arc we need to find a 0-length path from 3 to 7. Due to the release date and due date restrictions, there is only one path with that property shown in Fig. 14. Thus there is only one negative cycle and that cycle belongs to the spiral category.

Notice that for problem $P|r_j, p_j = 1|\sum w_j U_j$ with non-unit weights, the last arc terminating in $t = 7$ may have a positive weight. In particular, if in Example 5, condition $1 < w_7 < w_8$ holds and $d_7 = 8$ is replaced by $d_7 = 7$, then the spiral cycle shown in Fig. 14 has positive arc (6, 7) of length w_7. The cycle, however, is still negative; its length is $-w_8 + w_7$.

The algorithm for optimality check is presented in Appendix 3. Although the algorithm does not limit its search to spiral cycles, still it outperforms the fastest algorithm for finding the optimal solution for $P|r_j, p_j = 1|\sum w_j U_j$.

5 Problem $P|p_j = 1|\sum w_j U_j$

We consider now the special case when all jobs are available simultaneously ($r_j = 0$ for $j \in J$) and show that for this case it is sufficient to consider simpler cycles of chain type.

Theorem 5 *A compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule for problem $P|p_j = 1|\sum w_j U_j$, contains a negative cycle if and only if it contains a negative right-chain cycle.*

Proof Since problem $P|p_j = 1|\sum w_j U_j$ is a special case of $P|r_j, p_j = 1|\sum w_j U_j$, we can use the result of Theorem 4 and consider only negative spiral cycles. Let z be the left-most node of such a cycle. Consider a negative arc (t, τ) of the cycle, which corresponds to moving some job $j \in J_{(t,\tau)}$ to time interval $[\tau, \tau + 1[$. Since all release dates are zero, arc (t, τ) can be replaced by arc (t, z) which corresponds to moving the same job j to time interval $[z, z + 1[$, the length of the new arc being no larger than that of (t, τ). Since the remaining part of the spiral cycle from z to t is a right chain with 0-length arcs except for the last arc, which can be positive, the new cycle with the shortcut (t, z) is a right-chain cycle $t, z, \ldots, t$ of negative length. □

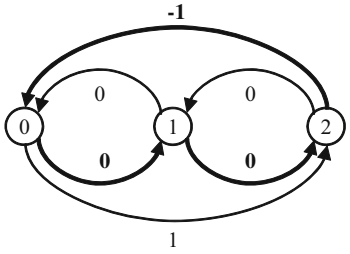


Fig. 15 A unique negative cycle for Example 6 belonging to the right-chain category

The result formulated in Theorem 5 cannot be strengthened for the unweighted version $P|p_j = 1|\sum U_j$, as the following example illustrates.

Example 6 Consider an instance of the problem $P|p_j = 1|\sum U_j$ with one machine and three jobs with due dates $d_1 = 2, d_2 = 3, d_3 = 1$, and a schedule given by job sequence (1, 2, 3). The corresponding graph, shown in Fig. 15, contains a unique negative cycle (2, 0, 1, 2) of length -1; it is a right-chain cycle; all two-node cycles are non-negative.

In what follows we show that network $\overline{N}(\mathbf{C})$ can be reduced by removing some nodes and arcs. Recall that the network reduction has been performed in Sect. 3 for problem $P|r_j, p_j = 1|\sum w_j C_j$ where it has been shown that transitive arcs are not needed.

We start with decomposing $\overline{N}(\mathbf{C})$ into so called zero-components. A *zero-component* $Z_i, 1 \le i \le \gamma$, consists of consecutive nodes $\underline{\tau}_i, \underline{\tau}_i + 1, \ldots, \overline{\tau}_i$, each of which can be reached from the start-node $\underline{\tau}_i$ via a chain of 0-length right arcs. For two zero-components Z_i and $Z_j, i < j$, there is no 0-length right arc (τ', τ'') connecting $\tau' \in Z_i$ and $\tau' \in Z_j$. Notice that Z_i may consist of a single node, i.e., $\underline{\tau}_i = \overline{\tau}_i$.

- Zero-component Z_i is the last one (i.e., $\overline{\tau}_i = \nu - 1$), if one of the jobs of Z_i has a due date no smaller than ν or if Z_i contains a late job; in both cases such a job can be moved to any later time slot at a zero cost, so that all right arcs originating from the corresponding node are of zero length.
- Zero-component Z_i is not the last one (i.e., $\overline{\tau}_i < \nu - 1$), if all jobs scheduled in the time-slots $\underline{\tau}_i, \underline{\tau}_i + 1, \ldots, \overline{\tau}_i$ are early and their maximum due date is equal to $\overline{\tau}_i + 1$.

The following example illustrates the decomposition of $\overline{N}(\mathbf{C})$ into zero-components.

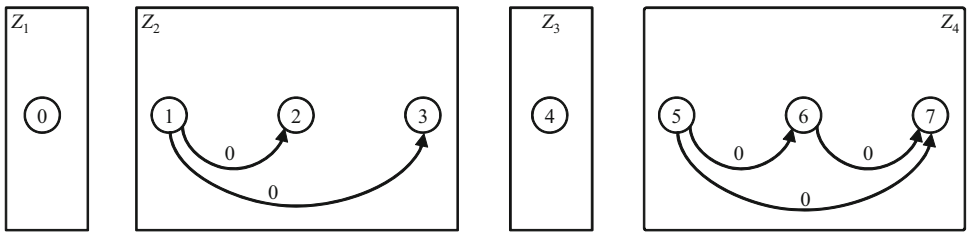


Fig. 16 Decomposition of $\overline{N}(\mathbf{C})$ for Example 7 into four zero-components

Example 7 Consider an instance of the problem $P|p_j = 1|\sum w_j U_j$ with $m = 3$ machines and $n = 24$ jobs with parameters given by the table:

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
d_j	1	1	1	2	3	4	3	3	3	4	4	5	5	5	5	6	7	8	1	2	5	8	8	8
w_j	3	7	9	5	9	6	2	5	5	9	8	11	6	7	8	9	5	3	10	4	3	6	8	5
C_j	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8
Node number	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7

In the table we separate job triples, which are processed in the same unit time slot, by vertical lines. A schedule with four zero-components marked by rectangles is shown in Fig. 16; the start- and end-nodes of the components are as follows:

$$\underline{\tau}_1 = 0;\ \overline{\tau}_1 = 0;$$
$$\underline{\tau}_2 = 1;\ \overline{\tau}_2 = 3;$$
$$\underline{\tau}_3 = 4;\ \overline{\tau}_3 = 4;$$
$$\underline{\tau}_4 = 5;\ \overline{\tau}_4 = 7.$$

Decomposition of a schedule into zero-components can be performed by scanning the nodes $\tau = 0, 1, \ldots, \nu - 1$ left to right, identifying for a current zero-component Z_i the furthest node from the start-node reachable by a chain of 0-length right arcs. We provide a formal description of procedure 'Decomposition' in Appendix 4 and show that its time complexity is $O(n)$.

Based on zero-components $Z_1, Z_2, \ldots, Z_\gamma$, we define the notion of an essential cycle and prove that only essential cycles need to be considered in the optimality conditions. For all zero-components Z_i except for the last one ($1 \le i \le \gamma - 1$) introduce characteristic w_i^* as the smallest w value among the jobs of that component:

$$w_i^* = \min\left\{w_j \mid \underline{\tau}_i + 1 \le C_j \le \overline{\tau}_i + 1\right\}. \tag{14}$$

Let τ_i^* be the node associated with w_i^*, i.e. $[\tau_i^*, \tau_i^* + 1[$ is the time slot where the job with weight w_i^* is scheduled. A right-chain cycle is *essential* if it is of the form $t, \underline{\tau}_i, \ldots, \tau_i^*, t$ and consists of

- a negative arc $(t, \underline{\tau}_i)$ with t belonging to the last component Z_γ and $\underline{\tau}_i$ being a start-node of the zero-component Z_i, $1 \le i \le \gamma$,
- a chain of right arcs of zero length from $\underline{\tau}_i$ to τ_i^*, all belonging to Z_i,
- a final arc (τ_i^*, t) terminating at t.

Notice that if $1 \le i < \gamma$, then the final arc (τ_i^*, t) is of positive length and if $i = \gamma$, then that arc is of zero length. Essential cycles of Example 7 are shown in Fig. 17.

Theorem 6 *Network $\overline{N}(\mathbf{C})$ contains a negative cycle if and only if it contains a negative essential cycle.*

Proof Due to Theorem 5 we can consider only negative right-chain cycles. Let t be the right-most node of such cycle and (t, τ) be its negative arc. This implies that time interval $[t, t + 1[$ contains a late job and therefore t belongs to the last zero-component Z_γ.

By the definition of a zero-component, any right arc from a node of one component to a node of another component is of positive length. Since there is no more than one positive arc in a right-chain cycle, the origin t is followed by the nodes of one zero-component only, say Z_i, $1 \le i \le \gamma$. Thus a negative right-chain cycle can be represented as $t, \tau_i', \ldots, \tau_i'', t$, where the zero length path $\tau_i', \ldots, \tau_i''$ consists of right arcs within Z_i. Notice that for a special case with $\tau_i' = \tau_i''$ we get a two-node cycle. We show that the length of the essential cycle $t, \underline{\tau}_i, \ldots, \tau_i^*, t$ is no larger than that of an arbitrary negative right-chain cycle $t, \tau_i', \ldots, \tau_i'', t$ with $\tau_i', \ldots, \tau_i''$ belonging to Z_i. Indeed, the left arcs of the two cycles satisfy $\ell_{(t,\underline{\tau}_i)} \le \ell_{(t,\tau_i')}$ since $\underline{\tau}_i < \tau_i'$; the internal paths within Z_i are of zero length for both cycles, i.e., $\ell_{(\underline{\tau}_i,\ldots,\tau_i^*)} = \ell_{(\tau_i',\ldots,\tau_i^*)} = 0$, while for the final arcs condition $\ell_{(\tau_i^*,t)} = w_i^* \le \ell_{(\tau_i'',t)}$ holds due to the definition (14) of w_i^*. □

We illustrate how the formulated optimality condition can be used for the optimality check in Appendix 4; the resulting algorithm has time complexity $O(n)$, an improvement in comparison with the $O(n \log n)$-time algorithm for finding an optimal schedule for problem $P|p_j = 1|\sum w_j U_j$ (Brucker and Kravchenko 2006; Dessouky et al. 1990).

6 Problem $P|r_j, p_j = 1|\sum w_j T_j$

Consider problem $P|r_j, p_j = 1|\sum w_j T_j$ with the weighted tardiness objective function. It appears that the negative cycles for problem $P|r_j, p_j = 1|\sum w_j T_j$ are more complicated than those introduced in Sect. 2.3 (two-node cycles, chain cycles or spiral cycles). In fact there may exist instances for which the simplest negative cycles have more than one negative arc - a major distinctions from the three types of cycles with one negative arc.

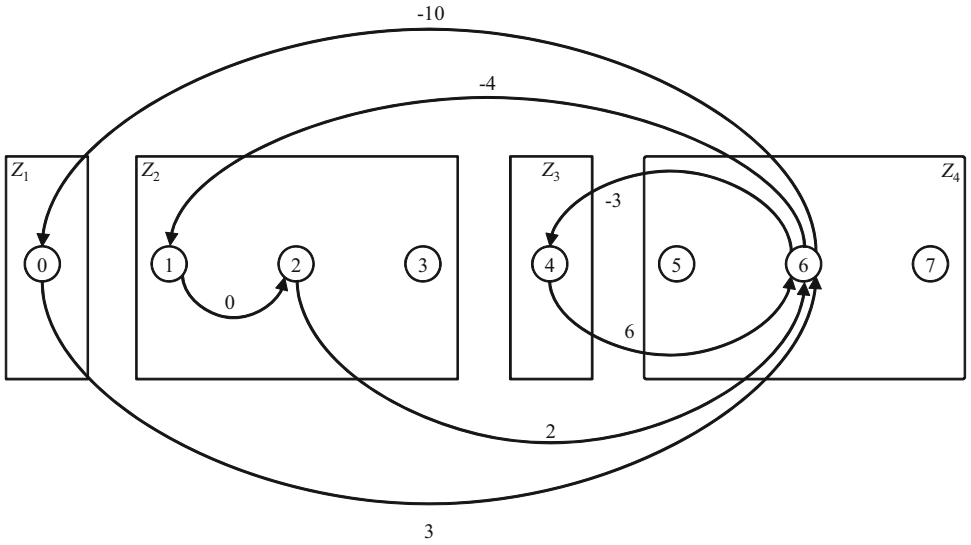


Fig. 17 Essential cycles of Example 7

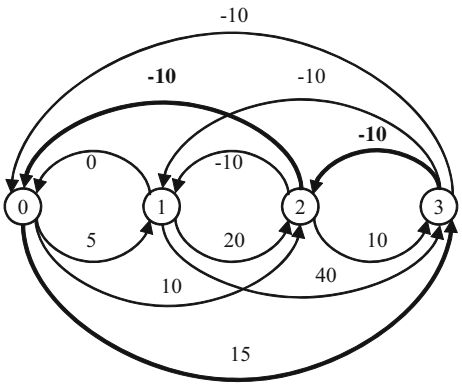


Fig. 18 A graph for Example 8 with the unique negative cycle containing two negative arcs

Example 8 Consider an instance of the problem $P|r_j, p_j = 1|\sum w_j T_j$ with one machine and $n = 4$ jobs given by the table

j	1	2	3	4
r_j	0	0	0	0
d_j	1	2	2	3
w_j	5	20	10	10
C_j	1	2	3	4

Figure 18 represents the corresponding graph; the unique negative cycle (3, 2, 0, 3) is marked by bold arcs and it contains two negative arcs (3, 2) and (2, 0).

Notice that the instance presented in Example 8 has zero release dates for all jobs, which implies that for problem $P|p_j = 1|\sum w_j T_j$ we also cannot limit our consideration to the three types of cycles defined above. In this paper we do not introduce new cycle types with several negative arcs; it is likely that the resulting rather complicated optimality conditions would have limited application.

In what follows we discuss how solution optimality can be checked using Theorem 1. Recall that Theorem 1 provides the necessary and sufficient optimality conditions for the most general problem $P|r_j, p_j = 1|\sum w_j f_j(C_j)$ with an arbitrary separable objective function. In order to verify the optimality of a given solution $\mathbf{C}$, we first apply the $O(n \log n)$-time Algorithm 'Left Shift($\mathbf{C}$).' Then the reduced graph $\overline{N}(\mathbf{C})$ is constructed in $O\left(\frac{n^2}{m}\right)$ time, as described in Sect. 2.3. It has $\nu = \lceil \frac{n}{m} \rceil$ nodes and a arcs, $a < \nu^2$. Since the fastest algorithms for negative cycle detection are of time complexity $O(\nu a)$ (Cherkassky and Goldberg 1999), which is $O\left(\frac{n^3}{m^3}\right)$ in our case, schedule optimality can be verified in $O\left(\frac{n^3}{m^3} + \frac{n^2}{m}\right)$ time - an improvement in comparison with the straightforward $O(n^3)$ approach based on finding an optimal solution to problem $P|r_j, p_j = 1|\sum w_j T_j$ via solving the associated assignment problem.

7 Problem $P|r_j, p_j = 1|\sum T_j$

We consider now the equal weight special case ($w_j = 1$ for $j \in J$) and show that for this case it is sufficient to consider simpler cycles of chain type.

Theorem 7 *A compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule for problem $P|r_j, p_j = 1|\sum T_j$, contains a negative cycle if and only if it contains a negative left-chain cycle.*

Proof Consider a negative cycle O with the smallest number of nodes. We prove that such a cycle satisfies Properties 1–4 of the definition of a spiral cycle and instead of Property 5, it satisfies a stronger property, namely

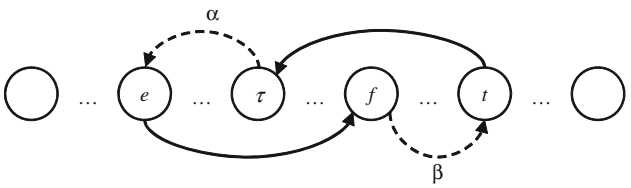


Fig. 19 Negative cycle for problem $P|r_j,\ p_j=1|\sum T_j$ that does not satisfy Property 5′

Property 5′: there is no right arc $(e,\ f)$ with the end-node f to the left of t.
Properties 1–4 combined with 5′ imply that a negative cycle with the smallest number of nodes is a left-chain cycle.

The proof of Properties 1–4 follows the same ideas as the proof of Theorem 4 for problem $P|r_j,\ p_j=1|\sum w_jU_j$; we present that proof in Appendix 5. Consider now Property 5′.

Let $(e,\ f)$ be the first right arc with $f<t$. We denote the fragment of cycle O from τ to e by α and the fragment of the schedule from f to t by β, see Fig. 19. Notice that by the definition of the spiral cycle

$$\begin{aligned} &e\le\tau<f, \qquad (15)\\ &\ell_\alpha=0,\\ &\ell_\beta\ge 0, \end{aligned}$$

see also Fig. 6.

Let j be a job in $J_{(t,\tau)}$ with the smallest due date, so that the length of arc (t,τ) is defined via that job,

$$\ell_{(t,\tau)}=-\left(t+1-\max\left\{d_j,\tau+1\right\}\right).$$

Notice that

$$d_j<t+1; \qquad (16)$$

otherwise the length of the arc (t,τ) is not negative. Similarly, let k be a job in J_e with the largest due date, so that the length of arc $(e,\ f)$ is defined via that job,

$$\ell_{(e,f)}=\max\left\{f+1-\max\left\{d_k,e+1\right\},\ 0\right\}.$$

Introduce an artificial right arc (e,t) and replace the fragment $(e,\ f),\beta$ of cycle O by a single arc (e,t). The length of the new cycle is

$$\begin{aligned}\ell_{(t,\tau)}+\ell_\alpha+\ell_{(e,t)}&=-\left(t+1-\max\left\{d_j,\tau+1\right\}\right)\\&\quad+0+\max\left\{t+1-\max\left\{d_k,e+1\right\},\ 0\right\}.\end{aligned}$$

Clearly, if $d_k>t+1$, then the last term in the above expression is 0, so that the resulting cycle is negative and it satisfies Property 5′. Also, if $\max\left\{d_j,\tau+1\right\}<d_k\le t+1$, then the resulting cycle is negative and it satisfies Property 5′ as well:

$$\begin{aligned}\ell_{(t,\tau)}+\ell_\alpha+\ell_{(e,t)}&=-\left(t+1-\max\left\{d_j,\tau+1\right\}\right)\\&\quad+0+\left(t+1-\max\left\{d_k,e+1\right\}\right)\\&=\max\left\{d_j,\tau+1\right\}-\max\left\{d_k,e+1\right\}\\&<d_k-\max\left\{d_k,e+1\right\}\le 0.\end{aligned}$$

Thus in what follows we assume

$$d_k\le\max\left\{d_j,\tau+1\right\}. \qquad (17)$$

We show that introducing an artificial left arc $(t,\ f)$ and replacing in the original cycle O the fragment $(t,\tau),\alpha,(e,\ f)$ by that arc we obtain a modified cycle M with less nodes which length ℓ_M is negative. To this end, we calculate the difference $\ell_M-\ell_O$ using the formula

$$\begin{aligned}\ell_M-\ell_O&=\left(\ell_{(t,f)}+\ell_\beta\right)-\left(\ell_{(t,\tau)}+\ell_\alpha+\ell_{(e,f)}+\ell_\beta\right)\\&=\ell_{(t,f)}-\ell_{(t,\tau)}-\ell_{(e,f)}.\end{aligned}$$

Let i be a job in $J_{(t,f)}$ with the smallest due date, so that the length of arc $(t,\ f)$ is defined via that job,

$$\ell_{(t,f)}=-\left(t+1-\max\left\{d_i,\ f+1\right\}\right).$$

Notice that $J_{(t,\tau)}\subseteq J_{(t,f)}$ and, as defined earlier for $J_{(t,\tau)}$, j is the job with the smallest due date among $J_{(t,\tau)}$. It follows that $d_i\le d_j$ so that

$$\ell_{(t,f)}\le-\left(t+1-\max\left\{d_j,\ f+1\right\}\right).$$

We consider the following cases with respect to d_j.

(1) The case of $d_j\ge t+1$ is eliminated due to (16).
(2) If $f+1\le d_j<t+1$, then

$$\begin{aligned}\ell_{(t,f)}&\le-\left(t+1-\max\left\{d_j,\ f+1\right\}\right)=-\left(t+1-d_j\right),\\-\ell_{(t,\tau)}&=t+1-d_j,\end{aligned}$$

and hence

$$\begin{aligned}\ell_M-\ell_O&\le-\left(t+1-d_j\right)+\left(t+1-d_j\right)-\ell_{(e,f)}\\&=-\ell_{(e,f)}\le 0.\end{aligned}$$

(3) If $\tau+1<d_j<f+1$, then due to $d_i\le d_j$,

$$\begin{aligned}\ell_{(t,f)}&=-\left(t-f\right),\\-\ell_{(t,\tau)}&=t+1-d_j,\\-\ell_{(e,f)}&=-\max\left\{f+1-\max\left\{d_k,e+1\right\},\ 0\right\}.\end{aligned}$$

Notice that

$$\max\left\{d_k,e+1\right\}\le\max\left\{d_j,\tau+1,e+1\right\}$$

$$= \max\{d_j, \tau + 1\} = d_j,$$

where we first use inequality (17), then (15), while the last equality holds due to the assumption of case (3). Hence

$$-\ell_{(e,f)} = -(f+1) + \max\{d_k, e+1\} \le -(f+1) + d_j.$$

We conclude that

$$\ell_M - \ell_O \le -(t-f) + (t+1-d_j) - (f+1) + d_j = 0.$$

(4) Finally, if $d_j \le \tau + 1$, then $d_k \le \tau + 1$ by (17) and

$$\begin{aligned} \ell_{(t,f)} &= -(t-f), \\ -\ell_{(t,\tau)} &= t - \tau, \\ -\ell_{(e,f)} &= -(f+1-\max\{d_k, e+1\}) \\ &\le -(f+1) + \max\{\tau+1, e+1\} \\ &= -(f+1) + (\tau+1) = \tau - f. \end{aligned}$$

Here we use (15) for $\max\{\tau+1, e+1\}$. It follows that

$$\ell_M - \ell_O \le -(t-f) + (t-\tau) + (\tau - f) = 0.$$

Thus Property 5′ is proved which, together with Properties 1–4 implies that the negative cycle with the smallest number of nodes belongs to the class of left-chain cycles. □

The algorithm to check the optimality of a given schedule is presented in Appendix 6. Its time complexity is $O(n \log n)$ provided that an earliest start schedule is given, and it reduces to $O(n)$ if all jobs are available simultaneously ($r_j = 0$ for all $j \in J$).

We now demonstrate that there are instances of problem $P|r_j, p_j = 1|\sum T_j$ for which negative left-chain cycles exist but there are no negative cycles of a simpler type, namely, two-node cycles.

Example 9 Consider an instance of the problem $P|r_j, p_j = 1|\sum T_j$ with $m = 1$ and $n = 4$ given by the table

j	1	2	3	4
r_j	0	0	1	2
d_j	4	2	3	3
C_j	1	2	3	4

Figure 20 represents a unique negative cycle for this instance which is a left-chain cycle. It is easy to verify that all two-node cycles are non-negative.

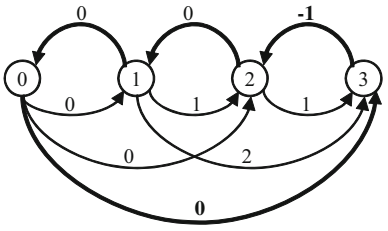


Fig. 20 A unique negative cycle for Example 9 of the left-chain type

8 Problem $P|p_j = 1|\sum T_j$

In problem $P|p_j = 1|\sum T_j$ all jobs are available simultaneously ($r_j = 0$, $j \in J$) and have unit weights. We first show that the necessary and sufficient conditions for problem $P|r_j, p_j = 1|\sum T_j$ can be simplified so that only two-node cycles are considered.

Theorem 8 *A compressed residual network $\overline{N}(\mathbf{C})$, representing a one-block earliest start schedule for problem $P|p_j = 1|\sum T_j$, contains a negative cycle if and only if it contains a negative two-node cycle.*

Proof Due to Theorem 7, if the graph for problem $P|r_j, p_j = 1|\sum T_j$ contains a negative cycle, it also contains a negative left-chain cycle. Given an arbitrary negative left-chain cycle we replace the whole left chain by a single arc. Since the length of that arc is no larger than that of the initial negative arc and the remaining removed left arcs are of zero lengths, we obtain a negative two-node cycle. □

Notice that unlike problem $P|r_j, p_j = 1|\sum w_j C_j$, the transitive arcs cannot be eliminated, as the following example shows.

Example 10 Consider an instance of the problem $P|p_j = 1|\sum T_j$ with one machine and three jobs with the due dates $d_1 = 2, d_2 = 2, d_3 = 1$. The schedule $\mathbf{C}$ is given by $C_1 = 1, C_2 = 2$ and $C_3 = 3$. If the transitive arcs $(2, 0)$ and $(0, 2)$ are not considered, then the two-node cycle consisting of arcs $(1, 0)$ and $(0, 1)$ is of length 0, another two-node cycle consisting of arcs $(2, 1)$ and $(1, 2)$ is of length $-1 + 1 = 0$. The two-node cycle consisting of transitive arcs $(2, 0)$ and $(0, 2)$ is, however, negative: $-2 + 1 = -1$.

Observe that the algorithm for optimality check for problem $P|p_j = 1|\sum T_j$ follows from the algorithm for the more general problem $P|r_j, p_j = 1|\sum T_j$ presented in Appendix 6. The time complexity of the latter algorithm is $O(n)$ for an earliest start schedule. Since in the case of $r_j = 0$ for all $j \in J$, Algorithm 'Left Shift($\mathbf{C}$)' transforms a given schedule into an earliest start schedule in $O(n)$ time, the optimality check for problem $P|p_j = 1|\sum T_j$ can be performed in $O(n)$ time for an arbitrary schedule.

9 Conclusions

In this paper we have studied the necessary and sufficient optimality conditions for various problems with parallel identical machines and unit job processing times. The formulated conditions deal with a specially defined network $\overline{N}(\mathbf{C})$ and they are based on detecting negative cycles. For each problem we give an insight into the underlying structure of the graph $\overline{N}(\mathbf{C})$ and specify the simplest types of cycles which need to be considered in the optimality conditions: two-node cycles, chain cycles and spiral cycles. The summary of the cycle types for different versions of problem $P|r_j, p_j = 1|F(\mathbf{C}, \mathbf{w})$ is provided in Table 1 in Sect. 2.3.

The special features of the cycles allow us to develop efficient algorithms for verifying optimality of a given schedule. We describe the ideas of the optimality check algorithms after formulating the optimality conditions for each problem and present all technical details in the appendices. The summary of the results is presented in Table 2 where we compare the time complexity of traditional algorithms for finding optimal schedules with the complexity of our optimality check algorithms.

In the table we assume that an earliest start schedule consisting of a single block is given (which can be verified in $O(n)$ time by the algorithm from Appendix 1). Otherwise an initial schedule can be converted into an earliest start schedule by Algorithm 'Left Shift' from Appendix 1; after that the optimality check can be performed for each block considered separately. Notice that Algorithm 'Left Shift' requires $O(n \log n)$ time if release dates are arbitrary and $O(n)$ time if all jobs are available simultaneously ($r_j = 0$ for all $j \in J$). This implies that 'Left Shift' does not affect the $O(n)$ time complexity of the algorithms for problems $P|p_j = 1|\sum w_j U_j$ and $P|p_j = 1|\sum T_j$; for other problems the time complexity should be increased by $O(n \log n)$ if the initial schedule does not belong to the class of earliest start schedules.

While we have illustrated only one application of the necessary and sufficient optimality conditions, they might be beneficial for solving other problems as well. Further applications of the optimality conditions may involve the development of new algorithms for finding alternative optimal solutions different from those produced by the known algorithms or algorithms for finding several optimal schedules. Another application area is hierarchical optimization, where it is required to select among solutions optimal for one criterion those solutions which minimize the secondary criterion. Finally, the optimality conditions play an important role in inverse scheduling (Heuberger 2004) where a target schedule is given and it is required to modify the problem parameters to make the target solution optimal. Developing optimality conditions for other scheduling problems and

Table 2 Time complexity of finding optimal solutions and performing the optimality check

Problem and cycle type of the optimality condition	Finding an optimal solution		Optimality check (for an earliest start schedule consisting of a single block)	
$P\|r_j, p_j = 1\|\sum w_j C_j$ two-node cycles	$O(n \log n)^*$		$O(n)$	Section 3
$P\|r_j, p_j = 1\|\sum w_j U_j$ spiral cycles	$O\left(\frac{n^2 \log m}{m} + n \log n\right)$	Dourado et al. (2009)	$O\left(\frac{n^2}{m} + n\right)$	Appendix 3
$P\|p_j = 1\|\sum w_j U_j$ right-chain cycles	$O(n \log n)$	Brucker and Kravchenko (2006); Dessouky et al. (1990)	$O(n)$	Appendix 4
$P\|r_j, p_j = 1\|\sum w_j T_j$ complex cycles	$O(n^3)$	Assignment problem	$O\left(\frac{n^3}{m^3} + \frac{n^2}{m}\right)$	Section 6
$P\|r_j, p_j = 1\|\sum T_j$ left-chain cycles	$O(n^3)$	Assignment problem	$O(n \log n)$	Appendix 6
$P\|p_j = 1\|\sum T_j$ two-node cycles	$O(n \log n)$	Dessouky et al. (1990)	$O(n)$	Appendix 6, Section 8

* Using a heap to select for each time slot available jobs with largest w_j

exploring their usage in the application areas listed above can be a subject of further research.

Acknowledgements This research was supported by the EPSRC funded project EP/D059518 "Inverse Optimization in Application to Scheduling."

Appendix 1: Earliest start schedule construction and verification

We present the pseudocodes of the algorithms described in Sects. 2.1, 2.2. The first algorithm constructs an earliest start schedule in $O(n \log n)$ time. The second algorithm verifies in $O(n)$ time whether a given schedule belongs to the class of earliest start schedules. The third one transforms in $O(n \log n)$ time a given schedule into an earliest start schedule, if it does not belong to this class.

Algorithm 'Calculate Blocks'

1. $i := 1; t := 0;$
2. While a job j exists with $r_j \geq t$ do
3. $\quad t := \min\{r_j | r_j \geq t\};$
4. $\quad t_i := t;$
5. $\quad B_i := \{j | r_j = t\};\ \ \mathcal{T}_i := \{t\};$
6. $\quad z := |B_i|;$
7. $\quad$ While $z \geq m + 1$ do
8. $\quad\quad$ Assign the first m jobs to time interval $[t, t + 1[;$
9. $\quad\quad h(t) := m;$
10. $\quad\quad t := t + 1;$
11. $\quad\quad B_i := B_i \cup \{j | r_j = t\};\ \ \mathcal{T}_i := \mathcal{T}_i \cup \{t\};$
12. $\quad\quad z := z + |\{j | r_j = t\}| - m;$
$\quad$ endwhile;
13. $\quad$ Assign all z jobs from B_i to time interval $[t, t + 1[$
14. $\quad h(t) := z;$
15. $\quad i := i + 1;$
16. $\quad t := t + 1$

endwhile

Algorithm 'Earliest Start Schedule Verification'

1. Scan unit time slots left to right identifying all time-values $\tau_1, \tau_2, \ldots, \tau_k$ with the property: $\tau_u = r_j$ for all jobs j scheduled in $[\tau_u, \tau_u + 1[$, $1 \leq u \leq k$; for completeness define τ_{k+1} as the makespan of the schedule (maximum job completion time); mark the identified τ values as 't-candidates'
2. If the first time slot of the schedule is not marked as a 't -candidate,' stop: property (i) of a block definition does not hold for the first time slot and the schedule is not an earliest start schedule;
3. Initialize $\rho := \tau_{k+1}$;
4. For $u = k$ to 1 by -1
5. $\quad \rho := \min\{\rho,\ \min\{r_j | \tau_u < S_j \leq \tau_{u+1} - 1\}\}$, where S_j is the starting time of job j;
6. $\quad$ If $\rho < \tau_u$, remove the mark of a 't-candidate' from τ_u;
7. endfor
8. If the first time slot of the schedule is not marked as a 't -candidate,' stop: property (ii) of a block definition does not hold for the first time slot and the schedule is not an earliest start schedule;
9. Define blocks $(B_i, \mathcal{T}_i)$ using 't-candidates' to specify block starting times;
10. For each block $(B_i, \mathcal{T}_i)$ do
11. $\quad$ If property (iii) of a block definition does not hold, stop: the schedule is not an earliest start schedule;
12. endfor
13. Output the t values for the earliest start schedule

Algorithm 'Left Shift(C)'

1. Calculate $\mathcal{T} = \cup_{i=1}^{r} \mathcal{T}_i$ and $h(t)$ for all $t \in \mathcal{T}$ using Algorithm 'Calculate Blocks';
2. Construct a list **H** of all jobs ordered according to non-decreasing r_j values; introduce pointers so that the position of job j in **H** and in **C** can be found in $O(1)$ time;
3. For each $t \in \mathcal{T}$ in increasing order do
4. $\quad$ In list **H**, mark all jobs j with $C_j = t+1$ as 'deleted';
5. $\quad \ell = |\{j | C_j = t + 1\}|$;
6. $\quad$ While $\ell < m$ and the first non-deleted job f in **H** satisfies condition $r_f \leq t$ do
7. $\quad\quad$ Move f in **C** to $[t, t + 1[$;
8. $\quad\quad \ell := \ell + 1;$
9. $\quad\quad$ In list **H**, mark f as 'deleted'
10. $\quad$ endwhile
11. endfor

The overall running time of Algorithm 'Left Shift(**C**)' is $O(n \log n)$. Indeed, the most time consuming step is the $O(n \log n)$-time Algorithm 'Calculate Blocks' needed to calculate $h(t)$. For efficient implementation of the remaining steps, an auxiliary data structure **H** is used to list the jobs in non-decreasing order of the release dates; in addition the pointers are introduced to locate jobs in list **H** and in the schedule **C** in constant time. In schedule **C**, moving a sin-

gle job requires constant time. In the list **H**, marking a job in **H** as 'deleted' requires $O(1)$ time, if pointers are used. Maintaining a special additional pointer and incrementing it throughout the algorithm, all steps of identifying the first non-deleted job f in **H** require $O(n)$ time.

Appendix 2: Verifying solution optimality for $P|r_j, p_j = 1|\sum w_j C_j$

Given an earliest start schedule, construct left arcs $A_{\text{left}}(\mathbf{C})$ of network $\overline{N}_{\text{TransRem}}(\mathbf{C})$ without transitive arcs, to complement them with the corresponding right arcs and to verify conditions (11) for $\nu - 1$ pairs of nodes t and τ. We first provide the details of the algorithm for constructing $A_{\text{left}}(\mathbf{C})$ and then analyze it, together with the remaining steps of the optimality check. As before, we assume that **C** consists of one block of $\nu = \lceil \frac{n}{m} \rceil$ unit time intervals.

Algorithm 'Construct Left Arcs $A_{\text{left}}(\mathbf{C})$ of $\overline{N}_{\text{TransRem}}(\mathbf{C})$'

1. For $t = 0$ to $\nu - 2$
2. Define $J_{(t+1,t)}$ as the set of jobs allocated to $[t+1, t+2[$ which can be moved to $[t, t+1[$;
3. If $J_{(t+1,t)} \neq \emptyset$ introduce left arc $(t+1, t)$ of length $\ell_{(t+1,t)} = -\max_{j \in J_{(t+1,t)}} \{w_j\}$;

endfor

4. If the resulting arcs connect all nodes into the single chain, then stop;

else

5. For each separate chain of left arcs, except for the last one,
6. Consider its right end-node τ and find the earliest time t from the next chain such that

$$J_{(t,\tau)} \neq \emptyset$$

(the set of jobs that can be moved from $[t, t+1[$ to $[\tau, \tau+1[$, $\tau < t$, is not empty);

7. Introduce left arc (t, τ) of length $\ell_{(t,\tau)} = -\max_{j \in J_{(t,\tau)}} \{w_j\} (t - \tau)$;

endfor

endif

Clearly, each for-loop of the algorithm scans the nodes of $\overline{N}_{\text{TransRem}}(\mathbf{C})$ once performing for each time slot t at most m operations on the jobs allocated to it. Thus the above algorithm can be implemented in $O(\nu m) = O(n)$ time.

For each left arc of the constructed out-tree, the length of its right arc counterpart can be found by (3). Since the number of left arcs in the out-tree is $\nu - 1$, the lengths of all right arcs of $\overline{N}_{\text{TransRem}}(\mathbf{C})$ can be found in $O(\nu m) = O(n)$ time as well.

Verifying conditions (11) for $\nu - 1$ pairs of nodes t and τ requires $O(\nu)$ time, if the lengths of arcs (t, τ) and (τ, t) have been determined. Thus the overall time complexity of the optimality check for problem $P|r_j, p_j = 1|\sum w_j C_j$ is $O(n)$. Notice that for problem $P|r_j, p_j = 1|\sum w_j C_j$, an optimal solution can be found in $O(n \log n)$ time by scheduling at each decision point an available job with the largest w_j value.

Appendix 3: Verifying solution optimality for $P|r_j, p_j = 1|\sum w_j U_j$

The fastest algorithm for solving problem $P|r_j, p_j = 1|\sum w_j U_j$ is due to Dourado et al. (2009) and it is of time complexity $O\left(\frac{n^2 \log m}{m} + n \log n\right)$. We show that the optimality check can be performed by a slightly faster algorithm of time complexity $O\left(\frac{n^2}{m} + n\right)$.

By Theorem 4, it is sufficient to explore whether there exists a negative spiral cycle for a given earliest start schedule. Our approach consists of three stages, where the first two stages deal with pre-processing. We assume that graph $\overline{N}(\mathbf{C})$ is constructed and its set of nodes corresponds to time intervals $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$.

Stage 1. Determine for each $\tau \in \mathcal{T}$ the left-most node $\lambda(\tau)$ reachable from τ via a 0-length left chain, and the right-most node $\mu(\tau)$ reachable from τ via a 0-length right chain.

Stage 2. Determine for each $\tau \in \mathcal{T}$ the left-most node $a(\tau)$ and the right-most node $b(\tau)$ reachable from τ via a 0-length path consisting of alternating left chains and right chains.

Stage 3. The negative arcs are considered one by one. For each arc (t, τ) of length $\ell_{(t,\tau)} < 0$, one of the two conditions happen: $a(\tau) \leq \tau < t \leq b(\tau)$ or $a(\tau) \leq \tau < b(\tau) < t$. In the former case there exists a negative cycle consisting of (t, τ) and a 0-length path from τ to t. In the latter case the algorithm finds the shortest path from τ to t as the smallest w value of one of the jobs processed in $[a(\tau), b(\tau)]$. We denote that value by $w_{[a(\tau),b(\tau)]}$,

$$w_{[a(\tau),b(\tau)]} = \min\left\{w_j | j \in J_z,\ a(\tau) \leq z \leq b(\tau)\right\}. \tag{18}$$

If $\ell_{(t,\tau)} + w_{[a(\tau),b(\tau)]} < 0$, then there exists a negative cycle consisting of the negative arc (t, τ), a 0-length path with the nodes belonging to $[a(\tau), b(\tau)]$ and a positive arc of length $w_{[a(\tau),b(\tau)]}$.

Graph $\overline{N}(\mathbf{C})$ can be constructed in $O\left(\frac{n^2}{m}\right)$ time, see Sect. 2.3. As we show below, the complexities of the subsequent three stages are $O\left(\frac{n^2}{m^2}+n\right)$, $O\left(\frac{n^2}{m^2}\right)$ and $O\left(\frac{n^2}{m^2}+n\right)$, respectively, so that the overall time complexity is $\left(\frac{n^2}{m}+n\right)$.

We start with the formal description of the procedure of Stage 1. It first determines for each $\tau \in \mathcal{T}$ the set of nodes $\mathcal{L}(\tau)$ to the left of τ connected with τ by 0-length left arcs. The values $\lambda(\tau)$ are defined one by one for $\tau = 0, 1, \ldots, \nu-1$. If $\lambda(0), \lambda(1), \ldots, \lambda(\tau-1)$ are known, then $\lambda(\tau)$ is the smallest value among $\lambda(\tau')$, where $\tau' \in \mathcal{L}(\tau)$, or $\lambda(\tau) = \tau$ if there are no 0-length left arcs originating from τ.

Finding μ values is similar, but there is one point of difference: the 0-length right arcs originating from τ are defined via early jobs from J_τ that can be moved to the right to the time slots no later than their due dates; however, if there is at least one late job J_τ, then all right arcs originating from τ are of 0 length. Formally, this procedure is described as follows.

Stage 1: 'Calculate $\lambda(\tau)$ and $\mu(\tau)$ for all $\tau \in T$'

1. For each τ from $\mathcal{T}=\{0, 1, \ldots, \nu-1\}$, set $r_\tau^{\min} := \min_{j\in J_\tau}\{r_j\}$, $d_\tau^{\max} := \max_{j\in J_\tau}\{d_j\}$,

$$\mathcal{L}(\tau) := \begin{cases} \emptyset, & \text{if } r(\tau) = \tau; \\ \{r_\tau^{\min}, r_\tau^{\min}+1, \ldots, \tau-1\}, & \text{otherwise;} \end{cases}$$

$$\mathcal{M}(\tau) := \begin{cases} \emptyset, & \text{if all jobs in } J_\tau \text{ have due date } \tau+1; \\ \{\tau+1, \tau+2, \ldots, d_\tau^{\max}-1\}, & \text{if all jobs in } J_\tau \text{ are early and } d_\tau^{\max} > \tau+1; \\ \{\tau+1, \tau+2, \ldots, \nu-1\}, & \text{if at least one job from } J_\tau \text{ is late;} \end{cases}$$

//λ-calculation:

2. $\lambda(0) := 0$;
3. For $\tau = 1$ to $\nu-1$ do
4. $\quad \lambda(\tau) := \min\left\{\tau,\ \min_{\tau'\in\mathcal{L}(\tau)}\left\{\lambda(\tau')\right\}\right\}$;
5. endfor

//μ-calculation:

6. $\mu(\nu-1) := \nu-1$;
7. For $\tau = \nu-2$ down to 0 do
8. $\quad \mu(\tau) := \max\left\{\tau,\ \max_{\tau'\in\mathcal{M}(\tau)}\left\{\mu(\tau')\right\}\right\}$;
9. endfor

Step 1 incurs $O(n)$ time, while the loops of Steps 3–5 and 7–9 require $O(\nu^2) = O\left(\frac{n^2}{m^2}\right)$ time.

The procedure for Stage 2 is based on the definition of a spiral cycle, see Sect. 2.3. The origin t of the negative left arc (t, τ) is not fixed, but node τ is fixed. The output for a given τ is the widest possible range of nodes $a(\tau), a(\tau)+1, \ldots, b(\tau)$ reachable via 0-length paths.

Stage 2: 'Calculate $a(\tau)$ and $b(\tau)$ for all $\tau \in T$'

1. For each τ from $\mathcal{T}=\{0, 1, \ldots, \nu-1\}$
2. $\quad$ Set $a(\tau) := \tau$ and $b(\tau) := \tau$;
3. $\quad$ While interval $[a(\tau), b(\tau)]$ can be extended do
4. $\quad\quad$ Identify the nodes of the left chain $L_1 = \{\lambda(\tau), \lambda(\tau)+1, \ldots, \tau\}$ originating from τ;
$\quad\quad$ Set $a(\tau) := \lambda(\tau)$;
$\quad\quad$ Scanning the nodes of L_1 and the associated μ values for them, determine the right-most node of the right chain R_1; set $b(\tau) := \max\{\mu(z) | z \in L_1\}$;
$\quad\quad$ Proceed in a similar manner with subsequent L_i and R_i, $i \geq 2$;
5. $\quad\quad$ endwhile
6. endfor

For a fixed τ, finding $a(\tau)$ and $b(\tau)$ incurs scanning no more than ν nodes. Hence the overall time complexity of finding $a(\tau)$ and $b(\tau)$ for all $\tau \in \mathcal{T}$ is $O(\nu^2) = O\left(\frac{n^2}{m^2}\right)$.

The key step of Stage 3 is calculating $w_{[\tau_1,\tau_2]}$ by (18) for all possible ranges $\tau_1 \leq \tau_2$. Clearly, if $w_{[\tau_1,\tau_2-1]}$ and $w_{[\tau_2,\tau_2]}$ are known, then

$$w_{[\tau_1,\tau_2]} = \min\left\{w_{[\tau_1,\tau_2-1]},\ w_{[\tau_2,\tau_2]}\right\}. \tag{19}$$

Thus the procedure of Stage 3 can be formulated as follows.

Stage 3 'Spiral Cycle Check'

1. For $\tau = 0$ to $\nu-1$
2. $\quad$ Define $w_{[\tau,\tau]} := \min_{j\in J_\tau}\left\{w_j\right\}$;
3. endfor
4. For $\tau_1 = 0$ to $\nu-2$
5. $\quad$ For $\tau_2 = \tau_1+1$ to $\nu-1$
6. $\quad\quad$ $w_{[\tau_1,\tau_2]} = \min\left\{w_{[\tau_1,\tau_2-1]},\ w_{[\tau_2,\tau_2]}\right\}$;
7. $\quad$ endfor
8. endfor
9. For each negative arc (t, τ) do
10. $\quad$ If $t \in [a(\tau), b(\tau)]$ stop: a negative cycle exists;
11. $\quad$ If $\ell_{(t,\tau)} + w_{[a(\tau),b(\tau)]} < 0$ stop: a negative cycle exists;
12. endfor

Steps 1–3 incur $O(\nu m) = O(n)$ time to calculate $w_{[\tau,\tau]}$, while Steps 4–8 incur $O(\nu^2)$ time to calculate $w_{[\tau_1,\tau_2]}$ for all $\tau_1 < \tau_2$. The total number of arcs (t,τ) examined in Steps 9–12 is $O(\nu^2) = O\left(\frac{n^2}{m^2}\right)$. For arc (t,τ), verifying conditions of Steps 10–11 takes $O(1)$ time. Thus the overall time complexity of Stage 3 is $O\left(\frac{n^2}{m^2}+n\right)$.

Appendix 4: Verifying solution optimality for $P|p_j = 1|\sum w_j U_j$

The fastest algorithm for solving problem $P|p_j = 1|\sum w_j U_j$ is of time complexity $O(n \log n)$, see Brucker and Kravchenko (2006), Dessouky et al. (1990). We show that the optimality check can be performed in $O(n)$ time based on Theorem 6.

First we notice that since in problem $P|p_j = 1|\sum w_j U_j$ all jobs are available simultaneously, algorithm 'Left Shift($\mathbf{C}$)' which transforms a given schedule into an earliest start schedule takes $O(n)$ time rather than $O(n \log n)$. In what follows we consider an earliest start schedule with time slots $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$.

Instead of constructing the whole graph $\overline{N}(\mathbf{C})$ with $O\left(\frac{n^2}{m^2}\right)$ arcs, we identify the zero-components $Z_1, Z_2, \ldots, Z_\gamma$ using procedure 'Decomposition' described below, and then perform the optimality check based on Theorem 6.

Decomposition is performed by considering the nodes $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$ left to right identifying for a current zero-component the furthest node from the start-node reachable by a chain of 0-length right arcs.

Procedure 'Decomposition'

1. Initiate the counter i for the zero-components as $i := 1$, set $\underline{\tau}_i = \overline{\tau}_i = 0$ and the current node $\tau = 0$;
2. While $\tau \le \overline{\tau}_i$ do
3. If there is a late job in J_τ, set $\overline{\tau}_i := \nu - 1$ to include in the zero-component all nodes to the right of τ and Stop;
4. Define the maximum due date $d_\tau^{\max}$ for the jobs J_τ scheduled in $[\tau, \tau+1[$:
$$d_\tau^{\max} = \max_{j \in J_\tau}\{d_j\};$$
5. If $d_\tau^{\max} > \overline{\tau}_i + 1$, then update $\overline{\tau}_i := \min\left\{d_\tau^{\max} - 1, \nu - 1\right\}$;
6. $\tau := \tau + 1$;
7. endwhile
8. Construction of Z_i is completed; If $\overline{\tau}_i < \nu - 1$, proceed with the next component: set $i := i + 1$, define new values of $\underline{\tau}_i, \overline{\tau}_i, \tau$, all equal to $\overline{\tau}_{i-1} + 1$, and perform Steps 2–6.

Notice that if condition of Step 5 holds, then the right arcs (τ, τ') are of zero length for each node $\tau' \in \{\tau + 1, \tau + 2, \ldots, d_\tau^{\max} - 1\}$ and hence all nodes τ' should be included in the current zero-component.

The time complexity of procedure 'Decomposition' is $O(n)$: it enumerates no more than $\nu = \left\lceil \frac{n}{m} \right\rceil$ nodes $\tau_i, 0 \le \tau \le \nu - 1$ performing each time $O(m)$ operations to check in Step 3 whether a late job in J_τ exists, or to find $d_\tau^{\max}$ in Step 4, and no more than one update of $\overline{\tau}_i$.

The algorithm for the optimality check enumerates essential cycles each of which starts in a node of the last zero-component Z_γ, reaches via a negative arc a start-node $\underline{\tau}_i$ of a zero-component Z_i, $1 \le i \le \gamma$, and returns back to the origin via right arcs. In order to perform the search efficiently, the algorithm uses auxiliary procedure 'Find Most Negative Arcs' which determines for each node $\tau \in \{0, 1, \ldots, \nu-1\}$ a negative arc $(T(\tau), \tau)$ with the origin $T(\tau)$ belonging to the last zero-component ($T(\tau) \in Z_\gamma$) such that the length of that arc, denoted by a negative value $-W(\tau)$ ($W(\tau) > 0$), has the most negative value among other arcs originating from $t \in Z_\gamma$:

$$W(\tau) = \max_{t \in Z_\gamma}\{w_j | j \in J_t^{\text{late}}, t \in Z_\gamma\}.$$

Here J_t^{late} is the set of late jobs scheduled in $[t, t+1[$, $J_t^{\text{late}} = \{j \in J_t \mid d_j \le t\}$. If there is no negative arc (t,τ) with the origin in Z_γ, we indicate this by setting $T(\tau) = 0$ and $W(\tau) = 0$. We describe the procedure for finding the values $T(\tau)$ and $W(\tau)$ after the main algorithm.

Suppose $T(\tau)$ and $W(\tau)$ are defined for all $\tau \in \{0, 1, \ldots, \nu - 1\}$. In the main algorithm we first verify whether there exists a negative essential cycle with all nodes belonging to the last zero-component Z_γ. This happens if there exists a negative arc leading to $\underline{\tau}_\gamma$, i.e., if $W(\underline{\tau}_\gamma) \ne 0$. The corresponding essential cycle consists then of the negative arc $(T(\underline{\tau}_\gamma), \underline{\tau}_\gamma)$ of length $-W(\underline{\tau}_\gamma) < 0$ and 0-length right arcs all belonging to Z_γ returning back to $T(\underline{\tau}_\gamma)$.

In the remaining part of the algorithm we consider all but last zero-components $Z_i, i = 1, \ldots, \gamma - 1$, and verify whether there exists a negative essential cycle passing via the start-node $\underline{\tau}_i$ of Z_i. To this end we identify the most negative arc $(T(\underline{\tau}_i), \underline{\tau}_i)$ of length $-W(\underline{\tau}_i)$ leading from a node of the last zero-component Z_γ to $\underline{\tau}_i$ and a positive arc of the smallest length w_i^* defined by (14) from a node of Z_i to Z_γ. If $-W(\underline{\tau}_i) + w_i^* < 0$, then the corresponding essential cycle consisting of $(T(\underline{\tau}_i), \underline{\tau}_i)$, 0-length right arcs belonging to Z_i and a positive arc returning back to $T(\underline{\tau}_i)$ is negative. Otherwise all essential cycles passing via $\underline{\tau}_i$ are non-negative since $(T(\underline{\tau}_i), \underline{\tau}_i)$ is the most negative arc leading to $\underline{\tau}_i$. Formally the main algorithm can be described as follows.

Algorithm 'Optimality Check for $P|p_j = 1|\sum w_j U_j$'

1. Use procedure 'Decomposition' to decompose a given schedule into zero-components $Z_1, Z_2, \ldots, Z_\gamma$;
2. Use procedure 'Find Most Negative Arcs' (formulated below) to determine for each $\tau \in \{0, 1, \ldots, \nu - 1\}$ the most negative arc $(T(\tau), \tau)$ of length $-W(\tau)$ with the origin $T(\tau)$ belonging to the last zero-component Z_γ;
3. If $-W(\underline{\tau}_\gamma) < 0$ then Stop: there exists an essential cycle with the negative arc $(T(\underline{\tau}_\gamma), \underline{\tau}_\gamma)$ and 0-length right arcs all belonging to Z_γ returning back to $T(\underline{\tau}_\gamma)$;
4. For $i = 1$ to $\gamma - 1$
5. Use (14) to calculate for the zero-component Z_i characteristic w_i^* as the smallest w value among all jobs of Z_i;
6. If $-W(\underline{\tau}_i) + w_i^* < 0$, then Stop: there exists an essential cycle with the negative arc $(T(\underline{\tau}_i), \underline{\tau}_i)$.
7. endfor

Step 1 involves procedure 'Decomposition' of time complexity $O(n)$. Step 2 uses the $O(n)$-time Procedure 'Find Most Negative Arcs' presented and analyzed below. Steps 3–7 are dominated by Step 5 which involves $O(\nu m) = O(n)$ jobs belonging to $Z_1 \cup Z_2 \cup \cdots \cup Z_{\gamma-1}$. Thus the overall time complexity for checking optimality of a solution to problem $P|p_j = 1|\sum w_j U_j$ is $O(n)$.

We now describe Procedure 'Find Most Negative Arcs' used in Step 2. It starts with setting up the initial values $W(\tau) := 0$ and $T(\tau) := 0$ for each $\tau = 0, 1, \ldots, \nu - 1$. In the first part of the algorithm (Steps 2–7) we identify negative arcs (t, τ) originating from all possible nodes $t \in Z_\gamma$ of the last zero-component by considering late jobs $j \in J_t^{\text{late}}$ of the corresponding time slot $[t, t + 1[$. A late job j induces negative arcs (t, τ), $(t, \tau - 1)$, …, $(t, 0)$ of the same length $-w_j$, where

$$\tau = d_j - 1. \tag{20}$$

For a fast implementation, we set up values $W(\tau)$ and $T(\tau)$ for nodes τ defined by (20) (Steps 2–7) and leave W and T values for $\tau - 1, \tau - 2, \ldots, 0$ to be updated in Steps 8–10. For those updates we use the property that for a negative arc (t, τ) there also exist negative arcs $(t, \tau - 1), \ldots, (t, 0)$ of the same length. We scan nodes τ right to left comparing $W(\tau - 1)$ and $W(\tau)$. Whenever $W(\tau - 1) < W(\tau)$, the updates are performed for $\tau - 1$ so that eventually we obtain for $\tau - 1$ the required values $T(\tau - 1)$ and $W(\tau - 1)$ which define the most negative arc for it.

Procedure 'Find Most Negative Arcs'

1. Initialize: set $W(\tau) := 0$ and $T(\tau) := 0$ for each $\tau = 0, 1, \ldots, \nu - 1$;
2. For each time slot $t \in Z_\gamma$ of the last zero-component do
3. For each late job $j \in J_t^{\text{late}}$ do
4. Define the largest τ such that the arc (t, τ) is negative: $\tau := d_j - 1$;
5. If $W(\tau) < w_j$, update $W(\tau) := w_j$ and $T(\tau) := t$;
6. endfor
7. endfor
8. For $\tau = \nu - 1$ down to 1 do
9. If $W(\tau - 1) < W(\tau)$, update $W(\tau - 1) := W(\tau)$ and $T(\tau - 1) = T(\tau)$;
10. endfor

Step 1 and Steps 8–10 deal with ν time-slots performing $O(1)$ operations for each of them. Steps 2–7 consider $O(n)$ jobs of $J_t^{\text{late}}, t \in Z_\gamma$, the number of updates for each job being $O(1)$. Thus the overall time-complexity of the procedure is $O(n)$.

Appendix 5: Spiral cycle properties for problem $P|r_j, p_j = 1|\sum T_j$

In this appendix we present a proof that for problem $P|r_j, p_j = 1|\sum T_j$, a negative cycle with the smallest number of nodes satisfies Properties 1–4 from the definition of a spiral cycle; the proof of the modified Property 5′, which is stronger than Property 5, is presented in Sect. 7.

Proof We use the same idea as in the proof of Theorem 4 for problem $P|r_j, p_j = 1|\sum w_j U_j$; the differences in the proofs are due to the formulae for calculating arc lengths which are problem specific.

The proof of Property 1 of Theorem 4 needs no changes up to formula (12): the length of the arc (e, f) should be replaced when considering objective $\sum T_j$ instead of $\sum w_j U_j$. For $\sum T_j$, the length of that arc is defined by job q which has the largest due date among the jobs in J_e:

$$d_q = \max_{j \in J_e} \{d_j\}$$

and the value is

$$\ell_{(e,f)} = \begin{cases} 0, & \text{if} \quad d_q \geq f + 1, \\ f + 1 - \max\{d_q, e + 1\}, & \text{otherwise.} \end{cases}$$

Depending on the value of d_q we have the following cases.

If $d_q \geq t_y + 1$, then $\ell_{(e,t_y)} = 0$ and we obtain a two-node negative cycle consisting of arcs (t_y, e) and (e, t_y). Notice that $\ell_{(t_y,e)} < 0$ since there is a late job in J_{t_y}. Alternatively, if $d_q < t_y + 1$, then

$$\ell_{(e,t_y)} = t_y + 1 - \max\left\{d_q, e + 1\right\},$$
$$\ell_{(t_y,f)} \leq f - t_y,$$

where the last condition may hold as inequality since there may exist a job in J_{t_y} with a due date larger than $t_y + 1$. Thus

$$\ell_{(e,t_y)} + \ell_{(t_y,f)} \leq f + 1 - \max\left\{d_q, e + 1\right\} = \ell_{(e,f)},$$

so that condition (13) holds and Property 1 is proved.

The proof of Properties 2-4 is the same as in Theorem 4. □

Appendix 6: Verifying solution optimality for $P|r_j, p_j = 1|\sum T_j$

Without loss of generality we assume that

$$r_j \leq d_j \leq \nu \tag{21}$$

for all jobs $j \in J$. Indeed, if $d_j > \nu$ for some job j, the value of d_j can be re-set to ν without affecting the objective value $\sum T_j$. If $d_j < r_j$ for some job j, the value of d_j can be re-set to r_j which decreases the objective value $\sum T_j$ by $r_j - d_j$, a constant that does not depend on the job sequence and does not affect the optimality of a schedule.

In order to formulate a fast algorithm for optimality check we first re-formulate the optimality conditions of Theorem 7 and then describe the optimality check algorithm.

Theorem 9 *For a negative left arc, let $J^{\text{late}}_{(t,\tau)}$ be a set of late jobs that can be moved from $[t, t+1[$ to $[\tau, \tau+1[$ and j^* be a job from that set with the smallest due date d_{j^*},*

$$d_{j^*} = \min\left\{d_j | j \in J^{\text{late}}_{(t,\tau)}\right\}.$$

Let $a(\tau)$ be the left-most node reachable from τ via a left chain $\tau, \ldots, a(\tau)$ of zero length and $d_{[a(\tau),\tau]}$ be the maximum due date among the jobs allocated to $a(\tau), a(\tau)+1, \ldots, \tau$,

$$d_{[a(\tau),\tau]} = \max_{a(\tau)\leq z\leq\tau}\left\{d_j | j \in J_z\right\}. \tag{22}$$

There exists a negative cycle with a negative left arc (t, τ) if and only if s

$$d_{[a(\tau),\tau]} > \max\{\tau + 1, d_{j^*}\}. \tag{23}$$

Proof Let j' be the job that delivers maximum in (22) and let $\tau', a(\tau) \leq \tau' \leq \tau$, be a time slot where job j' is scheduled, $j' \in J_{\tau'}$. Consider the left-chain cycle $(t, \tau, \ldots, \tau', t)$ consisting of the negative arc (t, τ), a 0-length left chain from τ to τ' followed by the right arc (τ', t). The components of that cycle have the lengths:

$$\ell_{(t,\tau)} = -\left(t + 1 - \max\left\{\tau + 1, d_{j^*}\right\}\right),$$
$$\ell_{(\tau,\ldots,\tau')} = 0,$$
$$\ell_{(\tau',t)} = t + 1 - \max\left\{\tau' + 1, d_{j'}\right\},$$

so that the length of the cycle is

$$\ell_{(t,\tau,\ldots,\tau',t)} = \max\left\{\tau + 1, d_{j^*}\right\} - \max\left\{\tau' + 1, d_{j'}\right\}.$$

If (23) holds, then

$$\max\{\tau + 1, d_{j^*}\} < d_{j'}$$

and hence

$$\ell_{(t,\tau,\ldots,\tau',t)} < d_{j'} - \max\left\{\tau' + 1, d_{j'}\right\} \leq 0,$$

i.e., (23) is a sufficient condition.

To prove that (23) is a necessary condition, suppose it does not hold for the negative arc (t, τ), i.e.,

$$d_{j'} \leq \max\{\tau+1, d_{j^*}\} \text{ for any } j' \in J_{\tau'}, a(\tau) \leq \tau' \leq \tau. \tag{24}$$

We show that all left-chain cycles are non-negative, which in combination with Theorem 7 implies that all possible cycles with arc (t, τ) are non-negative. Indeed, if $\tau' + 1 \leq d_{j'}$, then

$$\ell_{(t,\tau,\ldots,\tau',t)} = \max\left\{\tau + 1, d_{j^*}\right\} - d_{j'} \geq 0,$$

where the last inequality is by (24). Otherwise,

$$\begin{aligned}\ell_{(t,\tau,\ldots,\tau',t)} &= \max\left\{\tau + 1, d_{j^*}\right\} - \left(\tau' + 1\right)\\ &\geq (\tau + 1) - \left(\tau' + 1\right) \geq 0.\end{aligned}$$

□

The following corollary serves as the basis for our optimality check algorithm.

Corollary 1 *Let (t_1, τ) and (t_2, τ) be two negative arcs with the same end-node τ, and let j_1^* and j_2^* be two late jobs associated with these arcs, i.e., j_1^* and j_2^* are scheduled in $[t_1, t_1 + 1[$ and $[t_2, t_2 + 1[$, respectively, and they define the lengths of the corresponding arcs. If*

$$d_{j_1^*} \leq d_{j_2^*}$$

and $j^* = j_1^*$ does not satisfy the conditions of Theorem 9, then both left-chain cycles with arc (t_1, τ) and with arc (t_2, τ) are non-negative.

Proof Since condition (23) does not hold for $j^* = j_1^*$,

$$d_{[a(\tau),\tau]} \le \max\{\tau + 1, d_{j_1^*}\}.$$

Moreover, $\max\{\tau + 1, d_{j_1^*}\} \le \max\{\tau + 1, d_{j_2^*}\}$, so that condition (23) does not hold for $j^* = j_2^*$ as well. Hence by Theorem 9 no negative left-chain cycles exist with arc (t_1, τ) or with arc (t_2, τ). □

Consider a single-block earliest start schedule with $\nu = \lceil \frac{n}{m} \rceil$ unit time slots $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$. Corollary 1 provides the order in which negative arcs (t, τ) should be explored: late jobs $J^{\text{late}} = \{j | d_j < C_j\}$ should be considered in the non-decreasing order of their due dates, so that we can eliminate non-perspective late jobs if a job with a smaller due date does not satisfy (23). Let the jobs in J^{late} be renumbered in the non-decreasing order of due dates. The late job $j^* = 1$ has the smallest due date among J^{late} and t is its starting time, $t = C_{j^*} - 1$. All negative left arcs which lengths depend on d_{j^*} are of the form (t, τ) with the end-node τ belonging to the set $\mathcal{U}_{j^*} \cup \mathcal{V}_{j^*}$, where $\mathcal{U}_{j^*} = \{r_{j^*}, r_{j^*} + 1, \ldots, d_{j^*} - 1\}$ and $\mathcal{V}_{j^*} = \{d_{j^*}, d_{j^*} + 1, \ldots, t - 1\}$. Since

$$\max\{\tau + 1, d_{j^*}\} = \begin{cases} d_{j^*}, & \text{if} \quad \tau \in \mathcal{U}_{j^*}, \\ \tau + 1, & \text{if} \quad \tau \in \mathcal{V}_{j^*}, \end{cases}$$

condition (23) can be re-written as

$$d_{[a(\tau),\tau]} > d_{j^*} \quad \text{for} \quad \tau \in \mathcal{U}_{j^*}, \tag{25}$$

$$d_{[a(\tau),\tau]} > \tau + 1 \quad \text{for} \quad \tau \in \mathcal{V}_{j^*}. \tag{26}$$

If (25) does not hold for $\tau \in \mathcal{U}_{j^*}$ and (26) does not hold for $\tau \in \mathcal{V}_{j^*}$, then by Corollary 1 no negative left-chain cycle, originating from t or from another node, can pass via $\tau \in \mathcal{U}_{j^*} \cup \mathcal{V}_{j^*}$ and hence nodes $\mathcal{U}_{j^*} \cup \mathcal{V}_{j^*}$ can be excluded from further consideration. We perform a similar analysis with the next job $j^* = 2$ that has the second smallest due date among late jobs. The algorithm continues until all late jobs $j^* \in J^{\text{late}}$ are examined.

The formal description of the algorithm is presented below. We assume that the values $d_{[a(\tau),\tau]}$ are known for all τ, $0 \le \tau \le \nu - 1$; the procedure for finding these values is formulated later on.

Algorithm 'Optimality Check for $P|r_j, p_j = 1| \sum T_j$**'**

1. Initialize $\mathcal{T} := \{0, 1, \ldots, \nu - 1\}$;
2. Apply Procedure 'Calculate $d_{[a(\tau),\tau]}$ values for all $\tau \in \mathcal{T}$';
3. Renumber late jobs $J^{\text{late}} = \{j | d_j < C_j\}$ so that $d_1 \le d_2 \le \cdots \le d_z$, where $z = |J^{\text{late}}|$;
4. For $j^* = 1$ to z
5. Set $t := C_{j^*} - 1, \mathcal{U}_{j^*} := \{r_{j^*}, r_{j^*} + 1, \ldots, d_{j^*} - 1\}$ and $\mathcal{V}_{j^*} := \{d_{j^*}, d_{j^*} + 1, \ldots, t - 1\}$;
6. For $\tau \in \mathcal{T} \cap \mathcal{U}_{j^*}$ do
7. If $d_{[a(\tau),\tau]} > d_{j^*}$ stop: condition (25) holds for j^*;
8. endfor
9. For $\tau \in \mathcal{T} \cap \mathcal{V}_{j^*}$ do
10. If $d_{[a(\tau),\tau]} > \tau + 1$ stop: condition (26) holds for j^*;
11. endfor
12. Set $\mathcal{W}_{j^*} = \mathcal{U}_{j^*} \cup \mathcal{V}_{j^*}$ and $\mathcal{T} := \mathcal{T} \setminus \mathcal{W}_{j^*}$;
13. endfor

Notice that the set $\mathcal{U}_{j^*}$ may be empty; then condition (25) is not checked in Step 5.

We introduce and analyze the $O(n)$ Procedure 'Calculate $d_{[a(\tau),\tau]}$ values' later on. Here we explain how Steps 3–13 can be implemented in $O(n \log \frac{n}{m})$ time assuming that the initial schedule is the earliest start schedule. We also show that the time complexity reduces to $O(n)$ if all jobs are available simultaneously ($r_j = 0$ for all $j \in J$).

In Step 3 all values $\{d_j | j \in J^{\text{late}}\}$ belong to set $\mathcal{T} = \{0, 1, \ldots, \nu - 1\}$ due to assumption (21); therefore algorithm 'Distribution Counting' (Knuth 1998) can be used to sort the numbers in $O(n)$ time.

For efficient implementation of Steps 4–13 we maintain disjoint sets of prohibited τ-nodes which have already been eliminated, specifying each such set by its first and last nodes. Eliminating nodes $\mathcal{W}_1$ for job $j^* = 1$ results in the first prohibited set $I_1 = \mathcal{W}_1$. Eliminating nodes $\mathcal{W}_2$ for job $j^* = 2$ results in another prohibited set $I_2 = \mathcal{W}_2$ located to the right of I_1, if the left node of $\mathcal{W}_2$ is larger than the right node of I_1, or in an extended set I_1, otherwise. Notice that the case that the right node of $\mathcal{W}_2$ is smaller than the left node of I_1 does not happen: if it was a case, then the whole set $\mathcal{W}_2$ would be located to the left of $\mathcal{W}_1$, and hence two elements of those sets could not satisfy $d_1 \le d_2$, $d_1 \in \mathcal{W}_1$ and $d_2 \in \mathcal{W}_2$.

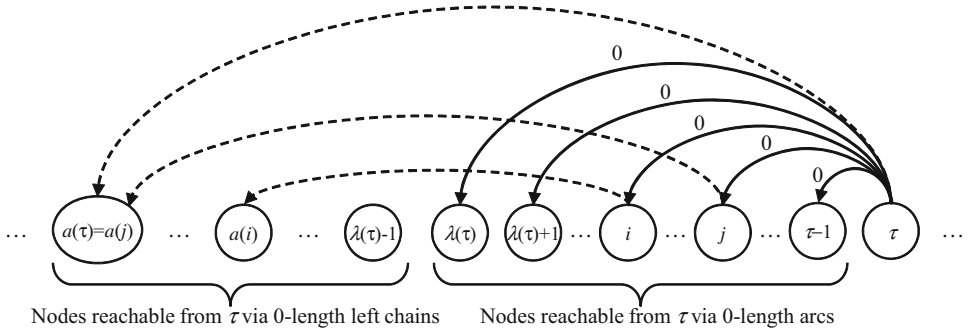
Suppose that after j iterations jobs $1, 2, \ldots, j$ have been examined and h disjoint prohibited intervals $I_1, I_2, \ldots, I_h$ have been produced, $h \le j$. It can be shown by induction that the most recent job j with the largest due date d_j belongs to I_h. Consider the next job $j + 1$ and the nodes $\mathcal{W}_{j+1}$ that should be eliminated. Eliminating nodes $\mathcal{W}_{j+1}$ results in

(i) another prohibited set $I_{h+1} = \mathcal{W}_{j+1}$ located to the right of I_h,
(ii) or in an extended set I_h.

The case that the right node of $\mathcal{W}_{j+1}$ is smaller than the left node of I_h could not happen: if it was a case, then elements $d_{j+1} \in \mathcal{W}_{j+1}$ and $d_j \in I_h$ of those sets could not satisfy $d_j \le d_{j+1}$.

During the course of the algorithm, no more than ν new disjoint sets I_h are created (in accordance with (i))

Fig. 21 Calculating d values



and extended (in accordance with (ii)). Creating a new set takes $O(1)$: it incurs only one comparison of the left end of $\mathcal{W}_{j+1}$ with the right end of I_h. Extending set I_h and its possible merger with several preceding sets involves finding the position of the left end of $\mathcal{W}_{j+1}$ with respect to $I_1, I_2, \ldots, I_h$, which can be done in $O(\log \nu)$ time, and creating one extended set, which can be done in $O(1)$ time. Thus the total number of updates associated with sets of eliminated nodes $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_z$ is $O(z \log \nu)$ or $O(n \log \nu)$ since $z \leq n$.

The loops of Steps 6–8 and 9–11 consider a separate τ value no more than once, so that the time complexity of those two loops is $O(\nu)$. Thus the overall time complexity of the optimality check is $O(n \log \frac{n}{m})$, or $O(n \log n)$ in the worst case, when the number of machines is $m = 1$.

Notice that if the jobs are available simultaneously ($r_j = 0$ for all $j \in J$), the time complexity reduces to $O(n)$ since all sets $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_z$ have the same left end 0, so that there is always only one prohibited interval I_1 with the left end at 0.

It remains to formulate the procedure that finds the values $d_{[a(\tau),\tau]}$ for all $\tau \in \mathcal{T}$. Suppose we have determined the values $d_{[a(i),i]}$ for all $i = 0, 1, \ldots, \tau - 1$ and we are calculating $d_{[a(\tau),\tau]}$. By (22), calculating $d_{[a(\tau),\tau]}$ involves enumerating the following nodes z, $a(\tau) \leq z \leq \tau$:

(a) node τ itself;
(b) each node i reachable from τ via a left arc (τ, i) of zero length;
(c) each node reachable from i via a 0-length left chain.

We illustrate the nodes that appear in (a)-(c) in Fig. 21. In that figure, solid arcs represent 0-length left arcs of type (b) and dotted arcs represent 0-length left chains of type (c). Notice that if $\tau' \in [a(\tau''), \tau'']$ for some τ'', then $a(\tau'')$ cannot lie to the right of $a(\tau')$ since there exists a 0-length chain from τ'' to τ' and then from τ' to $a(\tau')$. This implies

$$a(\tau'') \leq a(\tau') \leq \tau' \leq \tau'' \text{ for } \tau' \in [a(\tau''), \tau'']. \tag{27}$$

Furthermore, if $i \in [a(j), j]$ for some j, then $a(j)$ cannot lie to the right of $a(i)$ since there exists a 0-length chain from j to i and then from i to $a(i)$; this implies $[a(i), i] \subset [a(j), j]$.

If the set J_τ^{early} of early jobs scheduled in $[\tau, \tau + 1[$ is empty, then the nodes of type (b) and (c) do not exist. Otherwise all nodes τ' that satisfy (b) form the set $\{\lambda(\tau), \lambda(\tau) + 1, \ldots, \tau - 1\}$, where $\lambda(\tau)$ is the left-most node reachable from τ via a 0-length left arc,

$$\lambda(\tau) = \min_{j \in J_\tau^{\text{early}}} \{r_j\}.$$

Assuming that the values $d_{[a(i),i]}$ have been determined for all $i = 0, 1, \ldots, \tau - 1$, we can find $d_{[a(\tau),\tau]}$ by the formula:

$$d_{[a(\tau),\tau]} = \max \left\{ \max_{j \in J_\tau} \{d_j\}, \max_{i \in \{\lambda(\tau), \lambda(\tau)+1, \ldots, \tau-1\}} d_{[a(i),i]} \right\},$$

where the first term considers node τ of case (a) and the second term enumerates nodes $i \in \{\lambda(\tau), \lambda(\tau) + 1, \ldots, \tau - 1\}$ of types (b) and (c).

In the procedure presented below, $d_{[a(0),0]}$ receives the initial value $\max_{j \in J_0} \{d_j\}$, where J_0 is the set of jobs scheduled in $[0, 1[$. Each next value $d_{[a(\tau),\tau]}$ is pre-set initially to $\max_{j \in J_\tau} \{d_j\}$ and it is then adjusted by considering the values $d_{[a(i),i]}$ for $i = \tau - 1, \tau - 2, \ldots, \lambda(\tau)$. In our implementation, we do not enumerate all these i values explicitly. Instead, having examined $i = \tau - 1$ and the corresponding value $d_{[a(\tau-1),\tau-1]}$, we skip $i = \tau - 2, \tau - 3, \ldots, a(\tau - 1)$ since, by definition,

$$d_{[a(\tau-1),\tau-1]} = \max_{a(\tau-1) \leq z \leq \tau-1} \left\{ d_j | j \in J_z \right\},$$

and proceed with $i = a(\tau - 1) - 1$. Formally this approach can be formulated as follows.

Procedure 'Calculate $d_{[a(\tau),\tau]}$ values for all $\tau \in T$'

1. Set $\tau := 0$, $\lambda(\tau) := 0$ and $d_{[a(0),0]} := \max_{j\in J_0}\{d_j\}$;
2. For $\tau = 1$ to $\nu - 1$
3. Set $\lambda(\tau) := \min_{j\in J_\tau^{\text{early}}}\{r_j\}$, $d_{[a(\tau),\tau]} := \max_{j\in J_\tau}\{d_j\}$, $i := \tau - 1$;
4. While $i \geq \lambda(\tau)$ do
5. Update $d_{[a(\tau),\tau]} := \max\left\{d_{[a(\tau),\tau]}, d_{[a(i),i]}\right\}$;
6. Set $i := a(i) - 1$;
7. endwhile
8. endfor

Steps 1 and 3 consider $\tau = 0, 1, \ldots, \nu - 1$ and incur $O(\nu m) = O(n)$ time. In what follows we demonstrate that once the value $d_{[a(i^*),i^*]}$ is calculated for some $i = i^*$, that value may be involved in the subsequent calculations only once.

Consider the smallest value τ', $\tau' > i^*$, for which $i = i^*$ is used for updating $d_{[a(\tau'),\tau']}$ in Step 5. This happens if $i^* \in [a(\tau'), \tau']$ and there is a 0-length left arc (τ', i^*). The next attempt to use $i = i^*$ may happen whenever another node τ'' is considered, $\tau'' > \tau'$, for which there exists a 0-length left arc (τ'', τ'). Notice that by inequality (27),

$$a(\tau'') \leq a(\tau') \leq i^* \leq \tau' \leq \tau'' \text{ for } \tau' \in [a(\tau''), \tau''].$$

In Step 5, $d_{[a(\tau''),\tau'']}$ is updated with $d_{[a(\tau'),\tau']}$. By definition, $d_{[a(\tau'),\tau']}$ is the maximum due date among the jobs allocated to $\{a(\tau'), a(\tau') + 1, \ldots, i^*, \ldots \tau'\}$. Therefore, in Step 6 the i value is decreased from $i = \tau'$ to $i = a(\tau') - 1$ skipping all nodes $\{a(\tau'), a(\tau')+1, \ldots, i^*, \ldots \tau'\}$, including i^*.

Thus the total number of values $d_{[a(i),i]}$ involved in Step 5 in all iterations is $O(\nu)$, and the overall time complexity of Procedure 'Calculate $d_{[a(\tau),\tau]}$' is $O(n)$. This concludes the justification that the main algorithm for the optimality check is of time complexity $O(n \log n)$ and $O(n)$ for problems $P|r_j,\ p_j = 1|\sum T_j$ and $P|p_j = 1|\sum T_j$, respectively.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Englewood Cliffs, NJ: Prentice Hall.

Ahuja, R. K., & Orlin, J. B. (2001). Inverse optimization. *Operations Research*, *49*, 771–783.

Brucker, P., & Kravchenko, S. (2006). Scheduling equal processing time jobs to minimize the weighted number of late jobs. *Journal of Mathematical Modelling and Algorithms*, *2*, 245–252.

Cherkassky, B. V., & Goldberg, A. V. (1999). Negative-cycle detection algorithms. *Mathematical Programming*, *85*, 277–311.

Dessouky, M. I., Lageweg, B. J., Lenstra, J. K., & Van de Velde, S. L. (1990). Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, *44*, 115–123.

Dourado, M. C., Rodrigues, R. F., & Szwarcfiter, J. L. (2009). Scheduling unit time jobs with integer release dates to minimize the weighted number of tardy jobs. *Annals of Operations Research*, *169*, 81–91.

Heuberger, C. (2004). Inverse combinatorial optimization: A survey on problems, methods and results. *Journal of Combinatorial Optimization*, *8*, 329–361.

Knuth, D. (1998). *The Art of Computer Programming. Sorting and Searching* (Second ed., Vol. 3). Reading, MA: Addison-Wesley.

Lin, Y., & Wang, X. (2007). Necessary and sufficient conditions of optimality for some classical scheduling problems. *European Journal of Operational Research*, *176*, 809–818.