Polynomial algorithms for some scheduling problems with one nonrenewable resource

Abderrahim Sahli, Jacques Carlier, Aziz Moukrim

RAIRO-Oper. Res. 55 (2021) 3493–3511
https://doi.org/10.1051/ro/2021164

RAIRO Operations Research
www.rairo-ro.org

POLYNOMIAL ALGORITHMS FOR SOME SCHEDULING PROBLEMS WITH ONE NONRENEWABLE RESOURCE

Abderrahim Sahli[1,*], Jacques Carlier[2] and Aziz Moukrim[2]

Abstract. This paper deals with the Extended Resource Constrained Project Scheduling Problem (ERCPSP) which is defined by events, nonrenewable resources and precedence constraints between pairs of events. The availability of a resource is depleted and replenished at the occurrence times of a set of events. The decision problem of ERCPSP consists of determining whether an instance has a feasible schedule or not. When there is only one nonrenewable resource, this problem is equivalent to find a feasible schedule that minimizes the number of resource units initially required. It generalizes the maximum cumulative cost problem and the two-machine maximum completion time flow-shop problem. In this paper, we consider this problem with some specific precedence constraints: parallel chains, series-parallel and interval order precedence constraints. For the first two cases, polynomial algorithms based on a linear decomposition of chains are proposed. For the third case, a polynomial algorithm is introduced to solve it. The priority between events is defined using the properties of interval orders.

Mathematics Subject Classification. 90B35, 05C85.

Received February 7, 2021. Accepted October 26, 2021.

1. Introduction

In the literature, the Resource Constrained Project Scheduling Problem (RCPSP) plays a fundamental role in scheduling theory. In this problem, non-preemptive activities requiring *renewable* resources, and subject to precedence constraints, have to be scheduled in order to minimize the makespan. The RCPSP has led to an impressive amount of research in recent decades. A very effective way of solving this NP-hard problem is to decompose an RCPSP instance into as many Cumulative Scheduling Problems as there are resources. This allows us to obtain tight lower bounds as well as efficient head-tail adjustments [6,7]. The RCPSP with general time lag constraints has also been the subject of several papers [2, 8–10, 15, 19, 20]. Such works concern only renewable resources such as the workforce. Renewable resources are assigned to activities at their starting times and released at their completion times. On the contrary, *nonrenewable* resources are produced or consumed by activities at their starting times only. Money is an example of nonrenewable resource for which Carlier and Rinnooy Kan [3] introduced the financing problem.

Keywords. Scheduling problems, nonrenewable resource, decomposition method, series-parallel graph, interval order graph.

[1] COSYS/GRETTIA, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454 Marne-la-Vallée, France.

[2] Sorbonne universités, Université de Technologie de Compiègne, CNRS, laboratoire Heudiasyc UMR 7253, CS 60 319, 60 203 Compiègne Cedex, France.

*Corresponding author: abderrahim.sahli@esiee.fr

The Extended Resource Constrained Project Scheduling Problem (ERCPSP) is a general scheduling problem where the availability of resources is depleted and replenished [4]. An instance of ERCPSP is defined by events, nonrenewable resources and generalized precedence constraints between pairs of events. Each event produces or consumes some units of resources at its occurrence time. The objective is to build a schedule that satisfies the resource and precedence constraints and minimizes the makespan. ERCPSP is an extension of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources. In fact, we can associate with each instance of RCPSP an equivalent instance of ERCPSP [4,5]. Other authors have worked on models similar to ERCPSP. We can quote the works of Neumann and Schwindt [18] and of Laborie [14]. Neumann and Schwindt formalized the project scheduling problem with inventory constraints where the availability of each resource is at any time upper and lower bounded. To solve this problem, they introduced a branch-and-bound algorithm with a filtered beam search heuristic. Laborie [14] introduced the concept of a Resource Temporal Network (RTN). He proposed a constraint propagation algorithm to solve the problem.

The decision problem of ERCPSP consists of determining whether an instance has a feasible schedule or not. When there is only one nonrenewable resource, it is equivalent to find a feasible schedule that minimizes the number of resource units initially required. The maximum cumulative cost problem, which was shown NP-complete [24], is the special case where the events are sequenced on one machine in such a way that the maximum cumulative consumption is minimized. It corresponds to the problem investigated by authors of [1, 12, 24]. Abdel-wahab and Kameda [1] have considered the special case where the precedence constraints can be represented by a parallel chains graph and series-parallel graph. For the parallel chains case, they have introduced an algorithm for finding optimal schedules. The algorithm decomposes each chain into subchains then it provides an optimal schedule by sequencing these subchains. A dominant chain which minimizes the maximum cumulative cost is obtained. This algorithm has been generalized for the series-parallel case. Other authors worked on the Two-Machine Maximum Flow Time Problem with Series-Parallel Precedence Constraints. We can quote the works of Monma and Sidney [16, 17] and of Sekiguchi [23]. Flow shop scheduling is a type of scheduling where jobs need to be processed on a set of machines in identical order [25]. In [13], the authors deal with the two-machine flow shop scheduling problem with unlimited periodic and synchronized maintenance applied on both machines. Kaplan and Amir have introduced a simple method for determining the feasibility of the relocation problem [12]. They have proved that the relocation feasibility problem is equivalent to the two-machine flow-shop problem which can be solved in $O(n \log n)$ using the Johnson's rule [11]. The relocation problem can be considered as an ERCPSP problem with parallel chains precedence constraints where there are only two events in each chain. Note that the events in ERCPSP are not sequenced on one machine. So instead of sequencing events, we have to schedule them. Hence, more than one event can occur at the same time.

In this paper, we consider the decision problem of ERCPSP with one resource under some specific precedence constraints. We study three special cases which can be solved in polynomial time: the parallel chains case, the series-parallel case and the interval order case. A list algorithm is proposed to construct feasible schedules for the parallel chains case. This algorithm is based on a different and simple decomposition of chains. We also present another method of decomposition of chains which provides the same decomposition proposed in [1]. We show that these subchains can be seen as jobs of a flow-shop with two machines. Hence, a feasible schedule can be constructed by using the Johnson's algorithm. An adaptation of this algorithm is presented for the series-parallel case and for scheduling problem with cumulative continuous resources. Finally, a list algorithm is introduced for the interval order case to construct feasible schedules. The priorities of events are defined using the properties of interval orders.

The remaining of this paper is structured as follows. In Section 2 we formulate our problem. In Section 3 we show the relation between scheduling and sequencing problems. In Section 4 we investigate the decision problem of ERCPSP with one resource and parallel chains precedence constraints. In Section 5 we consider the decision problem of ERCPSP with one resource and series-parallel precedence constraints. In Section 6 we solve the decision problem of ERCPSP with one resource and interval order precedence graph, and finally we conclude this paper in Section 7.

2. Problem formulation

This paper deals with the Extended Resource Constrained Project Scheduling Problem (ERCPSP). An instance $I = (X, K, U, a, v)$ of ERCPSP consists of a set $X = \{0, 1, \ldots, n, n+1\}$ of events, a set K of nonrenewable resources, and a set U of precedence constraints.

The occurrence time of each event $i \in X$ is denoted S_i (also denoted $S(i)$). Of course S_i is not given and has to be determined. By convention, the two events 0 and $n+1$ are added to respectively define the start and the end of the schedule.

For each event $i \in X$, a_i^k represents the quantity of resource $k \in K$ produced or consumed by event i. If a_i^k is positive, then event i produces the quantity a_i^k of resource k, whereas if $a_i^k < 0$, it consumes the quantity $|a_i^k|$ of resource k. For each resource $k \in K$, $a_0^k = Q_k$ corresponds to the initial level of resource k. At any time, the resource availability must be positive or null for each resource $k \in K$.

The precedence constraints express relations of start-to-start between pairs of events. They have the form $S_i + v_{ij} \leq S_j$, where v_{ij} represents the time lag between events i and j. In this paper, we suppose that we have only positive time lags. So, if $(i, j) \in U$ then event j cannot occur before time $S_i + v_{ij}$.

A *schedule* $\mathcal{S}$ on event set X is a function assigning an occurrence time S_i to each event $i \in X$. The makespan of a schedule $\mathcal{S}$ can be computed as $C_{\max} = S_{n+1}$. A schedule is feasible if it satisfies the precedence constraints (2.1) and the resource constraints (2.2):

$$S_i + v_{ij} \leq S_j \qquad \forall (i, j) \in U \tag{2.1}$$

$$\sum_{i \in X(S,t)} a_i^k \geq 0 \qquad \forall k \in K, \forall t \in \{0, 1, \ldots, T\} \tag{2.2}$$

where $X(S, t) = \{i \in X \mid S_i \leq t\}$ is the set of events which have occurred by time $t \geq 0$, and T is some given upper bound on the makespan, which means that all events have to occur no latter than time T. An optimal schedule is a feasible schedule which minimizes the makespan.

In the following, we consider only the single-resource case of ERCPSP ($|K| = 1$). So, each instance will be defined by a quadruplet (X, U, a, v). The number of resource units produced or consumed by event i is a_i and the initial resource units of the project corresponds to a_0.

2.1. Decision problem

Let $I = (X, U, a, v)$ be an instance of ERCPSP. The Decision Problem consists of determining whether I has a feasible schedule or not. Solving this problem is equivalent to find a feasible schedule that minimizes the number of resource units initially required. In fact, let Q^* be the minimal number of resource units initially required to get a feasible schedule. If $Q^* \leq a_0$, then instance I is feasible. Otherwise, I is infeasible.

Example 2.1. Let $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ be the set of events. $a_0 = +5$, $a_1 = -1$, $a_2 = -2$, $a_3 = -1$, $a_4 = -1$, $a_5 = -1$, $a_6 = -1$, $a_7 = +1$, $a_8 = +1$, $a_9 = +1$, $a_{10} = 0$, $v_{01} = 1$ (event 1 must occur at least 1 time unit after event 0), $v_{0,2} = v_{0,3} = v_{1,4} = v_{1,5} = v_{2,6} = v_{3,6} = v_{4,7} = v_{5,7} = v_{6,8} = v_{7,9} = v_{8,10} = v_{9,10} = 1$.

The graph resulting from Example 2.1 is shown in Figure 1. The number associated with a node represents the number of resource units required for that event, and the number corresponding to an arc represents the time lag. The number corresponding to event 0 is equal to the initial number of resource units for the project.

The minimal number of resource units initially required to get a feasible schedule is $Q^* = 4$. It will be obtained by scheduling events $1, 4, 5, 7$ and 9 before events $2, 3$ and 6. Since $Q^* < a_0 = 5$, the considered instance is feasible.

An optimal schedule for this instance is $\mathcal{S}^* = \{S_0 = 0,\ S_1 = 1,\ S_2 = 1,\ S_3 = 3,\ S_4 = 2,\ S_5 = 2,\ S_6 = 4,\ S_7 = 3,\ S_8 = 5,\ S_9 = 4,\ S_{10} = 6\}$. Figure 2 shows the resource availability over time associated with $\mathcal{S}^*$. Note that we can verify that there is no schedule where event 6 precedes event 7.

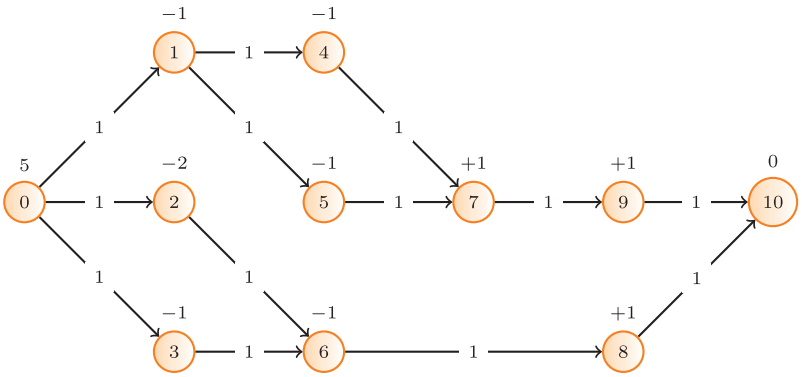

FIGURE 1. An instance of ERCPSP with seven events and one resource.

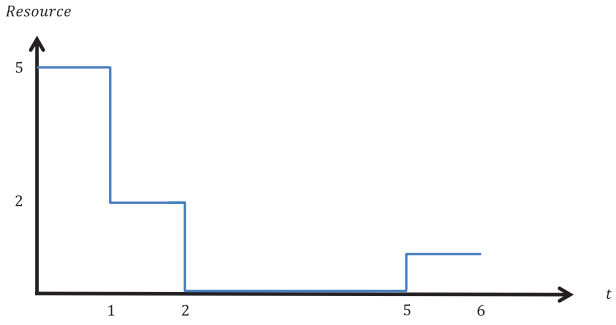

FIGURE 2. Resource availability curve associated with $\mathcal{S}^*$.

The optimisation problem of ERCPSP is NP-hard in the strong sense. In fact, Carlier *et al.* [5] have shown that the ERCPSP is an extension of the RCPSP which is NP-hard in the strong sense. The decision problem of ERCPSP is NP-complete. In fact, the special case where $a_i \in \{-1, +1\}$, $1 \leq i \leq n$, and $v_{ij} = 1$, $\forall (i,j) \in U$ generalizes the cumulative cost problem. This problem was proved to be NP-complete in [24].

2.2. Notations

Let $I = (X, U, a, v)$ be an instance of ERCPSP. We denote by $X^p = \{e \in X | a_e \geq 0\}$ (resp. $X^c = \{e \in X | a_e < 0\}$) the set of production (resp. consumption) events. We say that an event i is a *direct predecessor* of an event j if there exists a non-negative arc from i to j in the graph (X, U), which is equivalent to say that j is a *direct successor* of i. $l_{i,j}$ denotes the length of the longest path from i to j. We say that an event i is an *ascendant* of an event j if there exists a path from i to j with non-negative $l_{i,j}$, which is equivalent to say that j is a *descendant* of i. We denote the set of direct successors of an event i as $\Gamma^+(i)$, and the set of all descendants of i, not including i, as $\bar{\Gamma}^+(i)$. The corresponding sets of direct predecessors and ascendants are denoted respectively as $\Gamma^-(i)$ and $\bar{\Gamma}^-(i)$. Thus, event 0 (resp. event $n+1$) is an ascendant (resp. a descendant) of all the other events.

3. SEQUENCING AND SCHEDULING PROBLEMS

We consider a precedence graph with positive valuations. Let S be a feasible schedule for a value Q_0 of initial units of resource. We restrict ourselves to the case of a single resource. A sequence of events is said to be feasible, if the event order respects the precedence constraints and for each event e_i in the sequence, $Q_0 + a_{e_i}$ added to the sum of resource units produced and consumed by events before e_i is positive or null.

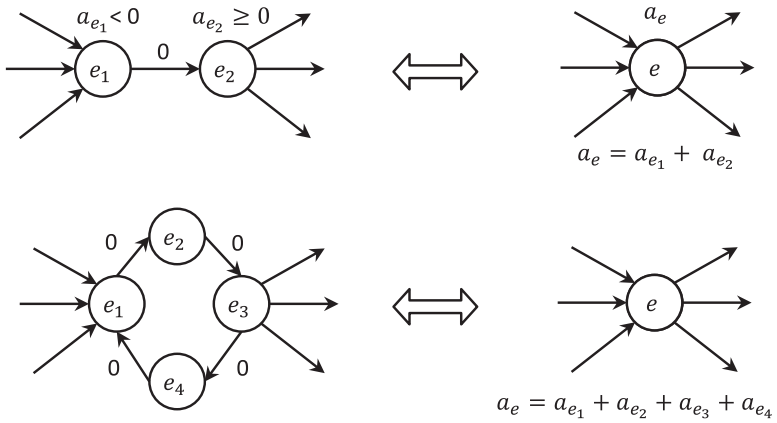

FIGURE 3. Pretreatment.

Theorem 3.1. *Let $I = (X, U, a, v)$ be an instance of ERCPSP. If all valuations are strictly positive then there is a feasible sequence of events as soon as there is a feasible schedule.*

Proof. Let S be a feasible schedule. If all the couples events are not executed simultaneously, the total order of events obtained by sequencing events in increasing order of their occurrence time is also feasible. Therefore, the only difficulty concerns the events which are executed at the same time in S. So let us consider a subset of events which are executed simultaneously in S. These events are independent (no precedence relations between them) because a strictly positive arc would prevent two independent events to be simultaneously executed. So we can sequence these events locally by ordering the production events before the consumption events. The result is a feasible sequence. By abusing the notation, we also denote by S the feasible sequence. □

One can generalize Theorem 3.1 in some cases of zero valuation by using the same reasoning.

Corollary 3.2. *If there are arcs with zero valuation, but none of them is between a consumption event and a production event, then there is an optimal sequence, as soon as there is an optimal schedule.*

Corollary 3.3. *If a consumption event is the only predecessor of a production event and the valuation of the corresponding arc is 0, it is dominant to execute these two events at the same time.*

So a pretreatment by fusion of such two events leads to the condition of Corollary 3.2. Similar fusion can treat the case of non-negative valuations with directed cycles of zero valuation (see Fig. 3).

4. The parallel chains case

In this section we investigate a special case of ERCPSP with one resource, where the precedence graph consists of a set of parallel chains with positive valuations. This special case is an extension of the problem considered by Abdel-wahab and Kameda [1], where more than one event can be executed at the same time. These authors introduced an algorithm for minimizing the maximum cumulative cost. The algorithm calculates the change of resource level, then determines production and consumption subchains in each chain. An optimal schedule is obtained by merging the production subchains in nondecreasing order of their rise, followed by consumption subchains in nonincreasing order of their fall. Thus, a dominant chain which minimizes the maximum cumulative cost is obtained.

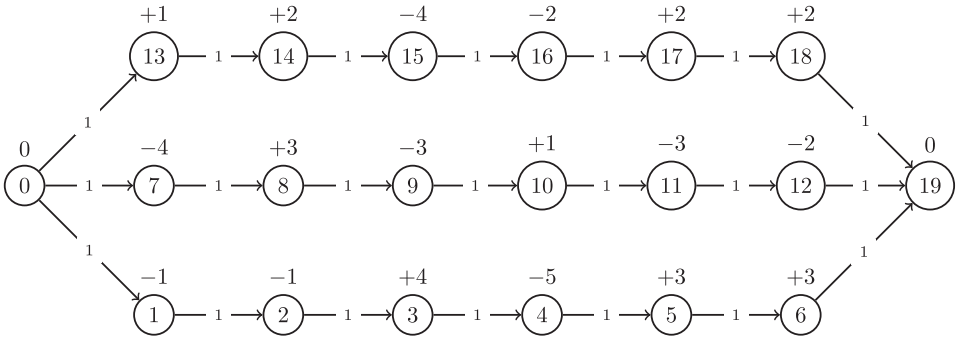

FIGURE 4. ERCPSP instance with three parallel chains.

In this section, we propose a list algorithm to construct feasible schedules for the parallel chains case. This algorithm is based on a different and simple decomposition of chains into production and consumption subchains. We also adapt the algorithm proposed in [1] to our problem. It always consists of determining the minimum amount of initial resources required. Abdel-wahab and Kameda sequenced the events because they are executed on one machine. In our problem there is no machine. Thus, some events can be executed at the same time. To use our algorithms, a pretreatment is required. For each consumption event ec and for each production event ep, if ec is the predecessor of ep and the valuation of the corresponding arc is 0, then we merge these two events into one event e such that $a_e = a_{ep} + a_{ec}$. After the pretreatment, we will be in the condition of Corollary 3.2. So finding a feasible schedule will be equivalent to finding a feasible sequence.

The algorithm is actually very similar. It is also based on a decomposition of chains into production and consumption subchains. We will see that these subchains can be seen as jobs of a flow-shop with two machines. The idea is to construct a standard schedule, where the events of each subchain are scheduled next to each other. Then, we apply the Johnson's rule to these subchains in order to obtain an optimal sequence. This method will be illustrated by the example given in Figure 4.

4.1. Definition of OP-subchains and OC-subchains

Suppose that chain h contains l events, $e_1, e_2, \ldots, e_l$, in the order of precedence constraints. Chain h will be decomposed into a sequence of optimal subchains. These optimal subchains can be seen as jobs of a flow-shop with two machines. Each subchain consists of two parts which respectively consumes and produces a quantity of resource. The consumption on the first part of the subchain corresponds to the processing time on the first machine, while the production on the second part of the subchain corresponds to the processing time on the second machine.

An optimal subchain is said to be an optimal production subchain (OP-subchain) if it produces more than it consumes. Otherwise, it is an optimal consumption subchain (OC-subchain).

Definition 4.1. Let $e_\alpha, \ldots, e_\gamma$ be a subchain of chain $e_1, e_2, \ldots, e_l$.

$e_\alpha, \ldots, e_\gamma$ is an optimal subchain if it can be decomposed into two subchains $e_\alpha, \ldots, e_\beta$ and $e_{\beta+1}, \ldots, e_\gamma$ such that:

- $\sum_{i=1}^{i=j} a_{e_i} \leq 0 \quad \forall j \in \{1, \ldots, \beta\}$.
- $\sum_{i=j}^{i=\beta} a_{e_i} \leq 0 \quad \forall j \in \{1, \ldots, \beta\}$.
- $\sum_{i=\beta+1}^{i=j} a_{e_i} \geq 0 \quad \forall j \in \{\beta+1, \ldots, \gamma\}$.
- $\sum_{i=j}^{i=\gamma} a_{e_i} \geq 0 \quad \forall j \in \{\beta+1, \ldots, \gamma\}$.

As a consequence, $\sum_{i=1}^{i=j} a_{e_i}$ reach its minimum, when j is equal to β. Furthermore, $\sum_{i=\beta+1}^{i=j} a_{e_i}$ will reach its maximum when $j = \gamma$. The fall Δ^- of an optimal subchain is equal to $-\sum_{i=1}^{i=\beta} a_{e_i}$, which is positive or null according to Definition 4.1. The fall corresponds to the processing time on the first machine of the flow-shop. The rise Δ^+ of an optimal subchain is equal to $\sum_{i=\beta+1}^{i=\gamma} a_{e_i}$, which is positive or null according to Definition 4.1. The rise corresponds to the processing time on the second machine of the flow-shop.

An optimal subchain can be a production subchain (OP-subchain) or a consumption subchain (OC-subchain). An OP-subchain is an optimal subchain such that $\Delta^- \leq \Delta^+$. An OC-subchain is an optimal subchain such that $\Delta^- > \Delta^+$. It is possible that the first part of an optimal subchain does not exist: β is not defined if the optimal subchain starts with a production event. It is also possible that the second part of an optimal subchain does not exist: $\beta = \gamma$.

Let us consider the example of Figure 4. The subchain $(1, 2, 3)$ is an OP-subchain: $e_\alpha = 1, e_\beta = 2, e_\gamma = 3, \Delta^- = -a_1 - a_2 = 2$ and $\Delta^+ = a_3 = 4$. The subchain $(7, 8, 9, 10)$ is an OC-subchain: $e_\alpha = 7, e_\beta = 9, e_\gamma = 10, \Delta^- = -a_7 - a_8 - a_9 = 4$ and $\Delta^+ = a_{10} = 1$. The subchain $(13, 14)$ is an OP-subchain where β does not exists. So, $\Delta^- = 0$ and $\Delta^+ = a_{13} + a_{14} = 3$

Theorem 4.2. *Given an optimal subchain, there exists a feasible sequence in which all events of this subchain are sequenced next to each other, as soon as a feasible schedule exists.*

Proof. Let us consider a feasible schedule of events. According to Theorem 3.1, we can deduce a feasible sequence of events S. Let $e_1, e_2, \ldots, e_l$ be a chain of events and $e_\alpha, \ldots, e_\gamma$ be one of its optimal subchains. Let e be an event not belonging to $\{e_1, \ldots, e_l\}$.

- If e is before e_α or after e_γ in S, then we do not move it.
- If e is the first event between e_α and e_β, then we shift e just before e_α. This respects the precedence constraints because there is no precedence relation between e and the events of $\{e_\alpha, \ldots, e_\beta\}$. It respects also the resource constraints because according to the definition of an optimal subchain $\sum_{i=\alpha}^{i=j} a_{e_i} \leq 0$ for each e_j before e in S.
- If e is the last event between $e_{\beta+1}$ and e_γ, then we shift e just after e_γ. This also respects precedence and resource constraints for the same reasons.

By iterating these modifications, the events of an optimal subchain will be consecutively sequenced. □

Definition 4.3. A sequence in which all events of each optimal subchain are ordered next to each other is said to be of standard form.

Corollary 4.4. *There exists a feasible sequence of standard form, as soon as there exists a feasible schedule.*

Proof. Given a feasible schedule, we deduce a feasible sequence S. Then, we successively apply the modifications described in the previous proof to the optimal subchains. We finally obtain a feasible sequence of standard form. We can remark that when the events of an OP-subchain (resp: OC-subchain) are already consecutive, they remain consecutive when applying the method to another OP-subchain (resp: OC-subchain). □

4.2. Decomposition of a chain into optimal subchains

In this section, we prove that one can decompose a chain into a subsequence of OP-subchains, followed by a subsequence of OC-subchains. For that, we report an algorithm to find the first and the shortest OP-subchain OP_1 of a chain. By removing OP_1 and iterating, we get the other OP-subchains. When there is no more OP-subchains, we can apply the same algorithm on the mirror chain of the remaining chain to get the OC-subchains. In fact, the definitions of OP-subchains and OC-subchains are perfectly symmetrical. Note that a chain $e'_1, \ldots, e'_l$ is said to be the mirror chain of $e_1, \ldots, e_l$ iff $\forall i \in 1, \ldots, l,\ a_{e'_i} = -a_{e_{l+1-i}}$.

Theorem 4.5. *A chain $e_1, \ldots, e_l$ can be decomposed into a subsequence of OP-subchains, followed by a subsequence of OC-subchains. We note:*

$$e_1, \ldots, e_l = \mathrm{OP}_1, \ldots, \mathrm{OP}_r, \mathrm{OC}_1, \ldots, \mathrm{OC}_s.$$

Proof. Algorithm 1 determines the shortest initial OP-subchain of a chain. By removing this OP-subchain and iterating, we get the other OP-subchains. When there are no more OP-subchains, we can apply the same algorithm on the mirror chain of the remaining chain to get the OC-subchains. Note that with this method of decomposition the sequence of the optimal subchains will not necessary respect the Johnson's rule (*i.e.* the fall (resp. rise) of the successive OP-subchains (resp. OC-subchains) monotonically increases (resp. decreases)). To do that, we have to search for the longest OP-subchain with minimal fall as it is explained in Section 4.4. □

Algorithm 1: Algorithm to determine the shortest initial OP-subchain.

$\alpha_1 \leftarrow 1$, $i \leftarrow 1$, $\beta_1 \leftarrow 0$, $\gamma_1 \leftarrow 0$, Sum $\leftarrow a_{e_i}$, Min $\leftarrow 0$;
while Sum < 0 **do**
  $i \leftarrow i + 1$
  if $i > l$ **then exit**; /* OP_1 does not exist */
  Sum $\leftarrow$ Sum $+ a_{e_i}$
  if Sum $<$ Min **then** $\beta \leftarrow i$; Min $\leftarrow$ Sum; $\mathrm{FALL}_1 \leftarrow -$Sum
  if Sum ≥ 0 **then** $\gamma_1 \leftarrow i$; $\mathrm{RISE}_1 \leftarrow$ Sum $-$ Min
end

4.3. List schedule for parallel chains case

We present here a method to solve the decision problem of ERCPSP with parallel chains precedence constraints. The idea is to decompose each chain into OP-subchains followed by OC-subchains which do not necessary respect the Johnson's rule. Then we use a list algorithm to construct a feasible schedule of standard form if the considered instance of ERCPSP admits solutions.

In the first phase of the algorithm, a priority is attributed to each optimal subchain. Based on this priority, the events which belong to the optimal subchain with the highest priority among the available subchains, are the first to be scheduled. The priorities of optimal subchains are defined as follows:

- Each OP-subchain has a higher priority than any OC-subchain.
- An OP-subchain OP_1 has a higher priority than an OP-subchain OP_2 iff the two OP-subchains are available and the fall of OP_1 is smaller than the fall of OP_2.
- An OC-subchain OC_1 has a higher priority than an OC-subchain OC_2 iff the two OC-subchains are available and the rise of OC_1 is larger than the rise of OC_2.

Theorem 4.6. *Algorithm 2 constructs a feasible schedule, as soon as the considered instance of ERCPSP with parallel chains precedence constraints admits any solution.*

Proof. Let I be an instance of ERCPSP with parallel chains precedence graph. Suppose that I is feasible and Algorithm 2 detects an over-consumption during the execution of an optimal production subchain OP_i. According to the algorithm, all the optimal subchains executed before OP_i are production subchains which provide resources. So, even if OP_i is executed before, an over-consumption will be detected. Moreover, according to the algorithm all the optimal subchains which can be scheduled instead of OP_i have a larger fall than OP_i (they need more resources to be scheduled). So we can deduce that if I is feasible then Algorithm 2 cannot detect an over-consumption during the execution of any OP-subchain. Symmetrically and similarly, we can prove that if I is feasible then this algorithm cannot detect an over-consumption during the execution of any OC-subchain. □

Algorithm 2: List algorithm to construct a feasible schedule.

```
Determine the OP-subchains of each chain using Algorithm 1
Determine the OC-subchains of each chain using Algorithm 1
pr ← total number of optimal subchains
while Some OP-subchains have not yet a priority do
    Let FP be the set of the first OP-subchains without priority of chains
    Attribute priority pr to the OP-subchain of FP having the smallest fall
    pr ← pr − 1
end
pr ← 0
while Some OC-subchains have not yet a priority do
    Let FC be the set of the last OC-subchains without priority of chains
    Attribute priority pr to the OC-subchain of FC having the smallest rise
    pr ← pr + 1
end
Let O be the set of all the optimal subchains
while |O| > 0 do
    Let o be the optimal subchain of O with the highest priority
    Schedule the events of o
    O ← O \ {o}
    if an over-consumption is detected then return("Infeasible instance")
end
return("Feasible instance")
```

4.4. Adaptation of the Johnson's rule

In this section, we present an adaptation of the Johnson's algorithm to solve the decision problem of ERCPSP with parallel chains precedence constraints. The idea is to determine the OP-subchains and OC-subchains of each parallel chain which respect the Johnson's rule: the fall (resp. the rise) monotonically increases (resp. decreases) for the successive OP-subchains (resp. OC-subchains). Then we construct a schedule of standard form that respects also the Johnson's rule. The obtained sequence minimizes the required amount of initial resources, so it is optimal.

Theorem 4.7. *A chain $e_1, \ldots, e_l$ can be decomposed into a subsequence of OP-subchains, followed by a subsequence of OC-subchains which respect the Johnson's rule: The fall monotonically increases for the successive OP-subchains and the rise monotonically decreases for the successive OC-subchains.*

Proof. Algorithm 3 determines the longest OP-subchain with minimal fall of a given chain. It can be iteratively used to find all the OP-, and OC-subchains in $O(l)$, where l is the number of events of the chain.

If $a_{e_1} \leq 0$, then the algorithm returns a subsequence of events $(e_\alpha, \ldots, e_\beta, \ldots, e_\gamma)$ which yields resources. Event e_β corresponds to the event giving the minimum value of SUM which represents the fall of OP_1. If $a_{e_1} > 0$, then e_β does not exist and the fall of OP_1 is equal to 0.

Algorithm 3 is defined so that the following minimum value of SUM is larger than the previous one. So, if OP_2 exists, then its fall is larger than the fall of OP_1. As a consequence, the fall of the successive OP-subchains monotonically increases. Hence, one of the conditions of the Johnson's rule is respected.

When we construct the OC-subchains using the mirror chain, we find that the rise of the successive OC-subchains monotonically decreases.

□

Algorithm 3: Algorithm to determine the initial longest OP-subchain.

$\alpha_1 \leftarrow 1$, $i \leftarrow 1$, $\beta_1 \leftarrow 0$, $\gamma_1 \leftarrow 0$, Sum $\leftarrow a_{e_i}$, Min $\leftarrow 0$
while Sum < 0 **do**
  $i \leftarrow i + 1$
  if $i > l$ **then exit**; /* OP_1 does not exist */
  Sum $\leftarrow$ Sum $+ a_{e_i}$
  if Sum $<$ Min **then** $\beta_1 \leftarrow i$, Min $\leftarrow$ Sum, $\mathrm{FALL}_1 \leftarrow -$Sum,
  if Sum ≥ 0 **then**
    $\gamma_1 \leftarrow i$, Max $\leftarrow$ Sum, $\mathrm{RISE}_1 \leftarrow$ Max $-$ Min
    $\alpha_2 \leftarrow i + 1$ /* Starting the construction of OP_2 */
  end
end
while Sum $\geq$ Min **do**
  $i \leftarrow i + 1$
  if $i > l$ **then** return $(e_{\alpha_1}, \ldots, e_{\gamma_1})$; /* $\mathrm{OP}_1 = (e_{\alpha_1}, \ldots, e_{\gamma_1})$ */
  Sum $\leftarrow$ Sum $+ a_{e_i}$
  if Sum $>$ Max **then**
    $\gamma_1 \leftarrow i$, Max $\leftarrow$ Sum, $\mathrm{RISE}_1 \leftarrow$ Max $-$ Min
    $\alpha_2 \leftarrow i + 1$ /* Starting the construction of OP_2 */
  end
end

Theorem 4.8. *There is an optimal sequence of standard form respecting the Johnson's rule, as soon as there exists a feasible schedule.*

Algorithm 4: Algorithm to construct an optimal sequence.

Determine the OP-subchains of each chain using Algorithm 3
Determine the OC-subchains of each chain using Algorithm 3
Schedule the OP-subchains sequentially in increasing order of their fall
Schedule the OC-subchains sequentially in decreasing order of their rise

Proof. Let us consider a sequence S of standard form respecting the Johnson's rule obtained by applying Algorithm 4. According to Corollary 4.4, we can deduce an optimal sequence S' of standard form from a given optimal schedule.

Now, suppose that an OP-subchain immediately succeeds an OC-subchain. We shift the OP-subchain immediately before the OC-subchain since the OP-subchain produces the resources and the OC-subchain consumes the resources. By iterating we obtain a new optimal sequence in which the OP-subchains are followed by the OC-subchains.

After that, we repeatedly interchange two adjacent OP-subchains, if the fall of the first OP-subchain is larger than the fall of the second one. Subsequently, we interchange two adjacent OC-subchains, if the rise of the first OC-subchain is smaller than the rise of the second one. By iterating we obtain a new optimal sequence S'' of standard form respecting the Johnson's rule.

Finally, we can deduce S from S'' by interchanging the adjacent OP-subchains (resp. OC-subchains) which have the same fall (resp. rise). So, Algorithm 4 constructs an optimal sequence that minimizes the required amount of initial resources. □

4.5. Application: continuous case

In ERCPSP, the action of producing or consuming resources is instantaneous. This assumption is generally made in the literature [26]. So, the level of resource is modified at some time point and remains constant till the

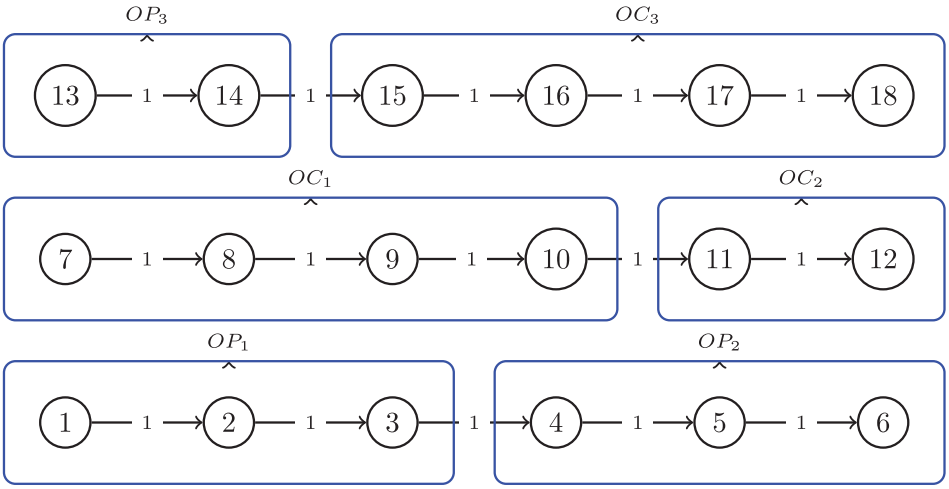

FIGURE 5. OP-, and OC-subchains of the example of Figure 4.

next discrete change (*i.e.* the level of resource is a step-wise function). However, this instantaneous production and consumption is inadequate for some real-world process scheduling problems. For example, the filling or the emptying of a tank by a liquid is usually subject to a constant rate depending of the number and the size of the siphons and taps. A second typical example is the load and unload of batteries.

This section addresses a scheduling problem with a cumulative continuous resource. Let Y be a set of preemptive tasks. Each task i has a processing time p_i and requires a continuously divisible resource during its processing time. The initial availability of this resource is equal to Q_0. We consider the case where the resource amount required by a task i at each time t is not fixed but is described by a continuous function. Let $b_i(\tau)$ be the resource requirement function of task i. Note that $b_i(\tau)$ is equal to the quantity of resource produced or consumed by i when its elapsed processing time is equal to τ.

Let $x_i(t)$ be a continuous function which determines whether a task i is in process or not at time t:

$$x_i(t) = \begin{cases} 1 & \text{if } i \text{ is in process} \\ 0 & \text{otherwise.} \end{cases}$$

The elapsed processing time of i at time t is given by $y_i(t)$:

$$y_i(t) = \int_0^t x_i(u)\,\mathrm{d}u.$$

We denote by $B_i(t)$ the cumulative production and consumption of task i at time t:

$$B_i(t) = \int_0^t x_i(u) b_i(y_i(u))\,\mathrm{d}u.$$

So the level of resource at time t is given by $\sum_{i\in Y} B_i(t) + Q_0$.

A schedule consists of determining for each task $i \in Y$ a function $x_i(t)$ which allows to know when task i is executed, *i.e.* we determine for each task $i \in Y$ the time intervals $[\delta_0^i, \delta_1^i], \ldots, [\delta_{d-1}^i, \delta_d^i]$ during which i is in process. Thus, we have:

$$x_i(t) = \begin{cases} 1 & \text{if } t \in [\delta_0^i, \delta_1^i] \cup \ldots \cup [\delta_{d-1}^i, \delta_d^i] \\ 0 & \text{otherwise.} \end{cases}$$

TABLE 1. An instance with 6 tasks.

Tasks	p_i	$b_i(x)$	$\bar{B}_i(x)$
1	3	$2x-2$	x^2-2x
2	3	$2x-4$	x^2-4x
3	3	$-2x+2$	$-x^2+2x$
4	3	$3x^2-9x+6$	$x^3-\frac{9}{2}x^2+6x$
5	5	$-3x^2+15x-12$	$-x^3+\frac{15}{2}x^2-12x$
6	5	$3x^2-15x+12$	$x^3-\frac{15}{2}x^2+12x$

A schedule is said to be feasible if $\sum_{i\in Y} B_i(t)+Q_0 \geq 0, \forall t \geq 0$. The objective in this problem is to construct a feasible schedule that minimizes the number Q_0 of resource units available initially.

To solve this problem, we start by decomposing each task into a sequence of subtasks that will be executed without preemption. These subtasks can be seen as jobs of a flow-shop with two machines. Each one of them consists of two parts that respectively consumes and produces a quantity of resource. The consumption of the first part corresponds to the processing time on the first machine, while the production of the second part corresponds to the processing time on the second machine. The subtasks of each task respect the Johnson's rule. Once the decomposition is done, the Johnson's rule is used to get an optimal sequence of subtasks that minimizes Q_0.

Each task i can be decomposed using Algorithm 3. Let us denote as $\bar{B}_i(t)$ the cumulative production and consumption of task i at time t, if it is continuously executed in $[0,p_i]$ (*i.e.* $\bar{B}_i(t)=\int_0^t b_i(u)\,\mathrm{d}u,\ \forall t\in[0,p_i]$). Let $t_0=0<t_1<\ldots<t_{l-1}<t_l=p_i$ be the critical points of $\bar{B}_i(t)$, *i.e.* $\bar{B}_i(t_0),\bar{B}_i(t_1),\ldots,\bar{B}_i(t_l)$ are the local extrema of $\bar{B}_i(t)$ in $[0,p_i]$. We associate with each task i a chain of events, where each event e_j corresponds to a critical point $t_j\in\{t_0,\ldots,t_l\}$ of $\bar{B}_i(t)$. The resource consumption or production of event e_j is defined as follows:

$$a_{e_j}=\begin{cases}0 & \text{if } j=0\\ \bar{B}_i(t_j)-\bar{B}_i(t_{j-1}) & \text{otherwise.}\end{cases}$$

There is an arc valued by t_j-t_{j-1} between each pair of events (e_{j-1},e_j). We decompose this chain into a sequence of optimal subchains using Algorithm 3. Each optimal subchain $e_\alpha,\ldots,e_\gamma$ defines a subtask of i whose processing time is equal to $t_\gamma-t_{\alpha-1}$ (we assume that $t_{\alpha-1}=0$ if $\alpha=0$). Its resource requirement, when the elapsed processing time is equal to τ, is given by $b_i(t_{\alpha-1}+\tau)$. If $e_\alpha,\ldots,e_\gamma$ is an OP-subchain (resp. OC-subchain), then the subtask is said to be an optimal production (resp. consumption) subtask and is denoted as $\mathrm{OP}_i(t_{\alpha-1},t_\gamma)$ (resp. $\mathrm{OC}_i(t_{\alpha-1},t_\gamma)$).

Example 4.9. Let us consider the instance provided in Table 1 which consists of 6 tasks $Y=\{1,2,3,4,5,6\}$.

The associated event chain and the optimal subtasks associated with each task of Example 4.9 are provided in Figure 6. For instance, Task 5 has 4 critical points $\{0,1,4,5\}$. So, it is associated with a chain of four events. The resource requirements of these events are respectively 0, $\bar{B}(1)-\bar{B}(0)=-5.5$, $\bar{B}(4)-\bar{B}(1)=13.5$ and $\bar{B}(5)-\bar{B}(4)=-5.5$. This chain consists of an OP-subchain of length 3 followed by an OC-subchain of length 1. Task 5 is therefore decomposed into optimal production subtask $\mathrm{OP}_5(0,4)$ followed by optimal consumption subtask $\mathrm{OC}_5(4,5)$. The processing times of these two subtasks are respectively equal to 4 and 1. Each subtask can be seen as a job of a flow-shop with two machines. For instance, the processing time on the first machine of $\mathrm{OP}_5(0,4)$ is equal to the fall of the associated subchain (*i.e.* it is equal to 5.5). Its processing time on the second machine is equal to the rise of the associated subchain (*i.e.* it is equal to 13.5).

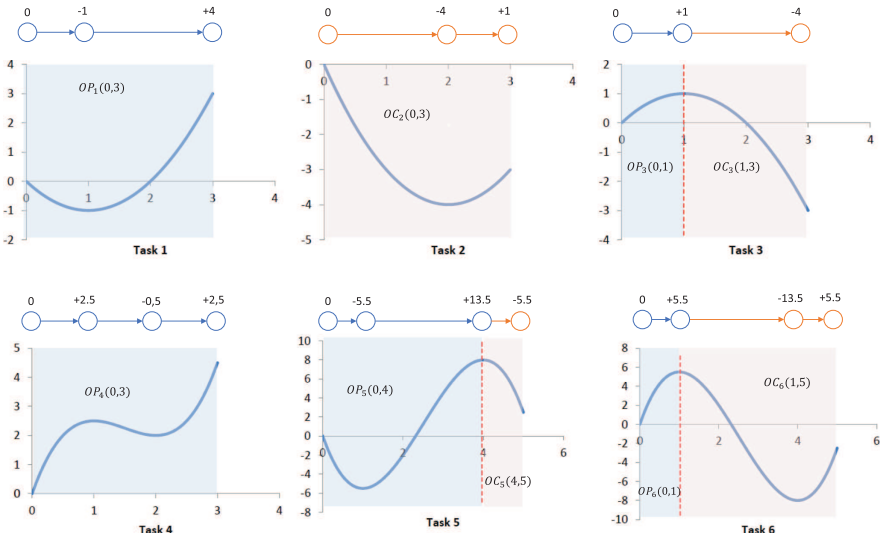

FIGURE 6. Decomposition of tasks.

Using the Johnson's rule, an optimal schedule can be obtained by sequencing the optimal subtasks as follows: $\mathrm{OP}_3(0,1), \mathrm{OP}_4(0,3), \mathrm{OP}_6(0,1), \mathrm{OP}_1(0,3), \mathrm{OP}_5(0,4), \mathrm{OC}_6(1,5), \mathrm{OC}_2(0,3), \mathrm{OC}_3(1,3), \mathrm{OC}_5(4,5)$.

5. The series-parallel case

We now consider a more general case, where the precedence relations involved can be represented by a series-parallel graph. This special case of ERCPSP is an extension of the problem considered by [1], where more than one event can be executed at the same time.

A series-parallel graph $G = (X, U)$ is a directed graph which can be obtained recursively from a single node by two operations, the *series composition* (Def. 5.1) and the *parallel composition* (Def. 5.2) of two series-parallel subgraphs [27].

Definition 5.1. Let $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ be two series-parallel graphs on disjoint sets. The series composition $G_s = (X_s, U_s)$ of G_1 and G_2 is defined as follows. $X_s = X_1 \cup X_2$ and $i \prec j \in U_s$ if and only if $i \prec j \in U_1 \cup U_2$, or $i \in X_1$ and $j \in X_2$. The sets X_1 and X_2 are called the *series blocks* of G_s.

Definition 5.2. Let $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ be two series-parallel graphs on disjoint sets. The parallel composition $G_p = (X_p, U_p)$ of G_1 and G_2 is defined as follows. $X_p = X_1 \cup X_2$ and $i \prec j \in U_p$ if and only if $i \prec j \in U_1 \cup U_2$. The sets X_1 and X_2 are called the *parallel blocks* of G_p.

Abdel-wahab and Kameda [1] define the series-parallel graphs as follows.

Definition 5.3 ([1])**.** A graph is a series-parallel graph if it can be reduced to a graph consisting of only two nodes with an arc between them by a sequence of the following operations.

(1) Replace two arcs, (u, v) and (v, w), and the node v by a single arc (u, w) if $|\Gamma^-(v)| = |\Gamma^+(v)| = 1$.
(2) Delete an arc in parallel to another arc.

From Definition 5.3, any series-parallel graph has a subgraph consisting of two chains (see Fig. 7), unless it is a single chain. If the precedence relations are represented by a series-parallel graph, then a total order of events can be defined as follows. We first find two parallel chains using the method proposed by [1]. Then we

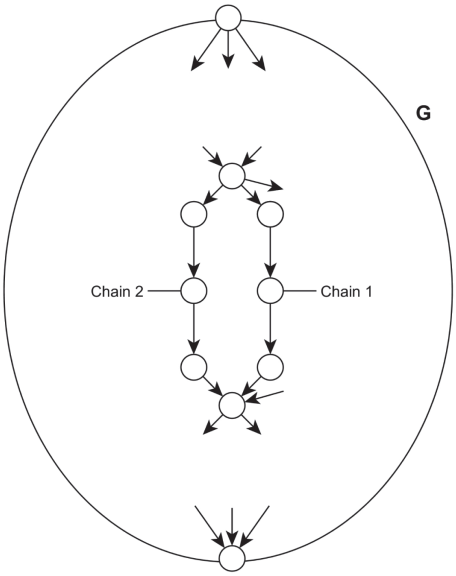

FIGURE 7. Series-parallel graph.

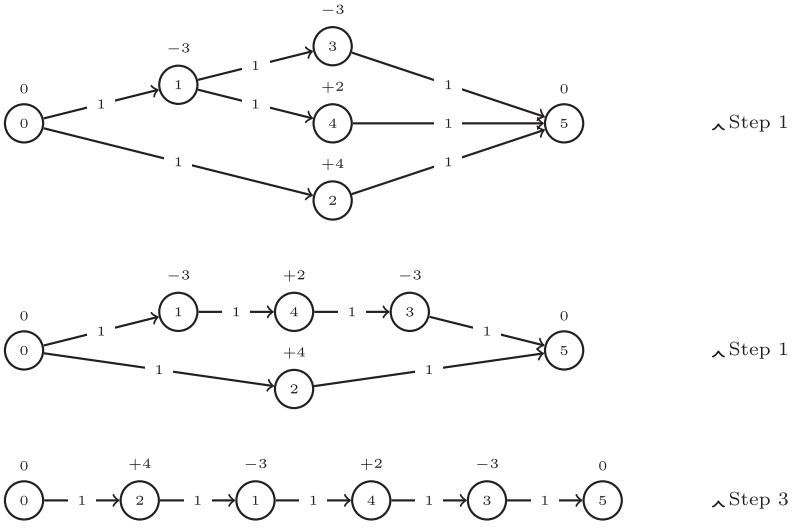

FIGURE 8. Example.

apply Algorithm 4 to obtain a locally optimal sequence. By reasoning as with previous section, we can show that this sequence is locally dominant. We replace the two parallel chains by a single chain obtained by adding an arc between two adjacent subchains according to the optimal sequence. Thus, we obtain another simpler series-parallel graph. By iterating the outcome is a single chain which corresponds to a total order of events. This method is illustrated by the example of Figure 8. Abdel-wahab and Kameda proved that any schedule, which respects this total order of events, minimizes the required amount of initial resources. The same proof can be used in the case of ERCPSP. It is based on the same principle as the case of parallel chains (The events

of each subchain are clustered around the pivot event, then the subchains are merged). Moreover, the feasibility of the problem in this case can be calculated using an $O(n^2)$ algorithm [1].

6. The interval order case

In this section, we investigate the special case of ERCPSP with one resource, where the precedence graph $G = (X, U)$ is an interval order graph and the time lags are strictly positive. We introduce for this special case a list algorithm to construct feasible schedules if any exists. The priorities of events are defined using the properties of interval orders, such that all production events are scheduled when they are ready, and all consumption events are scheduled when they are ready and are predecessors of all unscheduled production events.

6.1. Interval order graph

An interval order graph $G = (X, U)$ is a directed acyclic graph, such that for each $i \in X$, one can associate a closed interval $l(i)$ in the real line, such that for all $i, j \in X$, $(i, j) \in U$ if and only if $x < y$ for all $x \in l(i)$ and $y \in l(j)$ [22]. The system of intervals $l(i)$ is called an interval representation of G. Figure 9 provides an example of interval order graph and Figure 10 shows an interval representation of this example.

Proposition 6.1 ([21]). *Let (X, U) be an interval order graph. Then for $i, j \in X$, either all the successors of i are also successors of j, or all the successors of j are also successors of i.*

For any interval order graph, we can find a total order of elements $(i_0, i_1, \ldots, i_{n+1})$, such that $\Gamma^+(i_{n+1}) \subseteq \Gamma^+(i_n) \subseteq \ldots \subseteq \Gamma^+(i_0)$ (the successors of i_0 include the successors of i_1 which include the successors of $i_2, \ldots$, which include the successors of i_{n+1}). In the example of Figure 9, we have $\Gamma^+(1) = \{2, 4, 5, 7\}$, $\Gamma^+(2) = \{4, 7\}$ $\Gamma^+(3) = \{4, 7\}$, $\Gamma^+(4) = \{\}$, $\Gamma^+(5) = \{7\}$, $\Gamma^+(6) = \{4, 7\}$, $\Gamma^+(7) = \{\}$. We can note that $\Gamma^+(7) \subseteq \Gamma^+(4) \subseteq \Gamma^+(5) \subseteq \Gamma^+(6) \subseteq \Gamma^+(3) \subseteq \Gamma^+(2) \subseteq \Gamma^+(1)$. This property can be used to solve the decision problem of the interval order case. The idea is to schedule the production events as soon as possible, and the consumption events when they are available and respect the list $(i_0, i_1, \ldots, i_{n+1})$. An event is said to be available at time t if and only if all its predecessors are scheduled strictly before t.

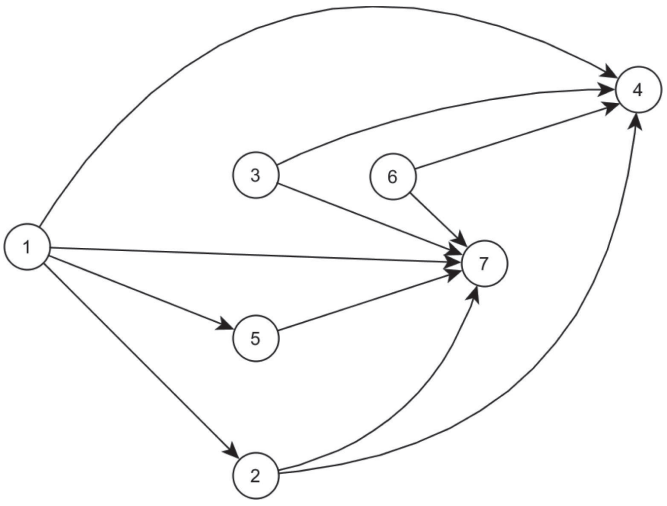

Figure 9. Interval order graph.

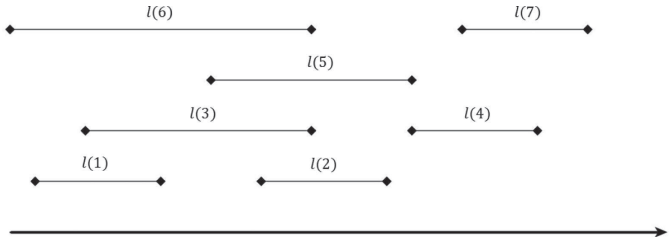

FIGURE 10. Interval representation.

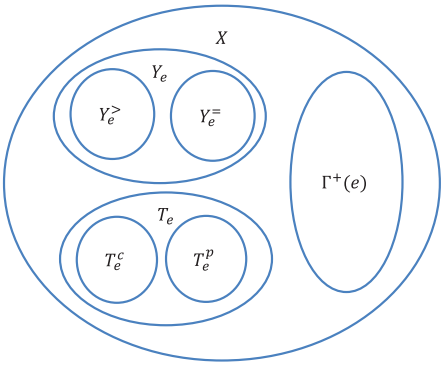

FIGURE 11. The subsets associated with event e.

6.2. List schedule for interval order case

Let $I = (X, U, a, v)$ be an ERCPSP instance with interval order precedence graph and strictly positive time lags. For each event $e \in X$, let Y_e be the subset which contains all the predecessors of all the successors of e. So, an event y belongs to Y_e if and only if for each event $e' \in \Gamma^+(e)$, y is a predecessor of e'. The subset Y_e can be partitioned into two subsets $Y_e^=$ and $Y_e^>$, where $Y_e^=$ contains the events which have the same successors than e and $Y_e^>$ contains the events which have more successors than e.

$$Y_e^> = \{e' \in X \mid \Gamma^+(e) \subset \Gamma^+(e')\}, \qquad Y_e^= = \{e' \in X \mid \Gamma^+(e) = \Gamma^+(e')\}$$

Let T_e be the subset which contains all the events not belonging to $Y_e \cup \Gamma^+(e)$. The subset T_e can be partitioned into two subsets T_e^p and T_e^c, where T_e^p contains the production events of T_e and T_e^c contains the consumption events. For each event $e' \in T_e$, e has more successors than e'. It follows that there is no precedence relation between each pair of events of $T_e \cup Y_e^=$. Figure 11 shows the different subsets associated with event e.

Algorithm 5 is a list algorithm which can be used to solve the decision problem of this special case of ERCPSP. In each iteration of the algorithm, all the available production events are scheduled first, followed by all the available consumption events which have the largest number of successors. If during some iteration the level of resource is negative, the algorithm increases enough the required initial resource units Q^* to satisfy the resource constraints. The algorithm terminates when a full schedule is constructed. We will show that at the end of the algorithm, the obtained schedule minimizes the required amount of initial resources. So if $Q^* \leq Q_0$, then the considered instance of ERCPSP is feasible.

Algorithm 5: Algorithm to solve the interval order case.

Let X^* be the set of events that are already scheduled;
$X^* \leftarrow \{0\}$;
$Q^* \leftarrow -\infty$; $t \leftarrow 0$;
while $X^* \neq X$ **do**
  while *some available production event ep is not yet scheduled* **do**
    `/* Schedule` ep `as soon as possible */`
    $S_{ep} \leftarrow \max(t, \max\{S_i + v_{i,ep} \mid i \in \Gamma^-(ep)\})$;
    $t \leftarrow S_{ep} + 1$;
    $X^* \leftarrow X^* \cup \{ep\}$;
  end
  if *some available consumption events are not yet scheduled* **then**
    Let ec be the one with the largest number of successors;
    `/* Schedule all the consumption events of` $Y_{ec}^{=}$ `*/`
    $t \leftarrow \max(t, \max\{S_i + v_{i,j} \mid j \in Y_{ec}^{=} \cap X^c \text{ and } i \in \Gamma^-(j)\}\})$
    for *all* $i \in Y_{ec}^{=} \cap X^c$ **do** $S_i \leftarrow t$
    $t \leftarrow t + 1$;
    $X^* \leftarrow X^* \cup Y_{ec}^{=}$ // $X^* = Y_{ec} \cup T_{ec}^{p}$
  end
  if $\sum_{e \in X^* \setminus \{0\}} a_e + Q^* < 0$ **then**
    `/* Increase the required amount of initial resources */`
    $Q^* \leftarrow -\sum_{e \in X^* \setminus \{0\}} a_e$
  end
end

Proposition 6.2. *If a consumption event ec is executed at time t by Algorithm 5, then all the events scheduled before or at time t are the ones belonging to* $Y_{ec} \cup T_{ec}^{p}$.

Proof. The consumption events are scheduled by Algorithm 5 in decreasing order of the number of their successors. Let ec_1 and ec_2 be two consumption events scheduled respectively at t_1 and t_2 such that $t_1 < t_2$. We suppose without loss of generality that between t_1 and t_2 only production events are processed. The first scheduled production event was not available at time t_1. So ec_1 is one of its predecessors. By iterating we show that ec_1 is a predecessor of all the production events scheduled between ec_1 and ec_2. Concerning ec_2, if it was available at time t_1, its priority is strictly inferior to the one of ec_1. So ec_1 has more successors than ec_2. Consequently ec_1 belongs to $Y_{ec_2}^{>}$. Otherwise if ec_2 was not available at time t_1, then a production event ep scheduled before t_2 by the algorithm precedes ec_2 and by transitivity ec_1 precedes ec_2. So ec_1 has more successors than ec_2. Consequently ec_1 belongs to $Y_{ec_2}^{>}$. As a result, if a consumption event ec is executed at time t by Algorithm 5, then all the consumption events of $Y_{ec}^{>}$ are executed before t and no event of T_{ec}^{c} is executed before t. Moreover, according to Algorithm 5 all the consumption events of $Y_{ec}^{=}$ are scheduled at time t.

Now we show that all the production events scheduled after ec are successors of ec. In fact, let ec_1, ec_2 and ec_3 be three consumption events scheduled at different and increasing times. All the production events scheduled strictly after ec_2 and before ec_3, are successors of ec_2. So they are also successors of ec_1 because the successors of ec_1 include the successors of ec_2. As a result, all the production events scheduled after ec are successors of ec. From this it follows that all the events of T_{ec}^{p} and all the production events of Y_{ec} are scheduled before or at time t. □

Theorem 6.3. *Let I be an ERCPSP instance with interval order precedence graph and strictly positive time lags. I is feasible if and only if for each event* $e \in X$ *we have* $\sum_{i \in Y_e \cup T_e^p} a_i \geq 0$.

Proof. This condition is necessary. In fact, let us consider a feasible schedule S and a time t when the first event e_1 of $\Gamma^+(e)$ is executed. All the predecessors of e_1 are necessary executed strictly before t. All the events

belonging to Y_e are predecessors of e_1 and each predecessor of e_1 not belonging to Y_e belongs to T_e. Let T'_e be the set of all the events of T_e scheduled strictly before t. Hence the events which are executed strictly before t are the ones belonging to $Y_e \cup T'_e$.

Since S is feasible, the level of resource at time $t - \epsilon$ must be positive. So we have

$$\sum_{i \in Y_e} a_i + \sum_{i \in T'_e} a_i \geq 0. \tag{6.1}$$

It is clear that

$$\sum_{i \in T^p_e} a_i \geq \sum_{i \in T'_e} a_i. \tag{6.2}$$

From (6.1) and (6.2) it follows that $\sum_{i \in Y_e \cup T^p_e} a_i \geq 0$.

Now we show that this condition is sufficient by studying the properties of Algorithm 5. Let S be the schedule obtained by using Algorithm 5. It is easy to verify that S is a time-feasible schedule. In fact, all the events are scheduled respecting all the precedence constraints. Moreover, suppose that a consumption event ec is scheduled by the algorithm at time t. According to Proposition 6.2, all the events scheduled before or at time t are the ones belonging to $Y_{ec} \cup T^p_{ec}$. So, if the condition of the theorem is true, the algorithm will construct a feasible schedule. Otherwise if for some consumption event ec, $\sum_{i \in Y_{ec} \cup T^p_{ec}} a_i$ is negative, the algorithm will increase enough the required initial resource units to satisfy the necessary condition of the theorem. This shows that at the end of the algorithm, we obtain a time-feasible schedule that minimizes the required amount of initial resources. □

7. Conclusion

In this paper we have considered the ERCPSP which is a general scheduling problem where the availability of resources is depleted and replenished. We have introduced the decision problem of ERCPSP and we have reported some complexity results. The decision problem in the general case is NP-complete, however some specific cases can be solved in polynomial time. We have presented three polynomial cases which are the parallel chains case, the series-parallel case and the interval order case. Of course, these algorithms cannot be applied directly to the general case because the problem is NP-hard. It is necessary to adapt them, for instance, by factoring arcs or suppression of arcs. This adaptation is the perspective of our work.

Acknowledgements. The authors would like to thank the anonymous reviewers whose comments have improved the quality of this article.

References

[1] H. Abdel-wahab and T. Kameda, Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Oper. Res.* **26** (1978) 141–158.

[2] M. Bartusch, R. Möhring and F. Radermacher, Scheduling project network with resource constraints and time windows. *Ann. Oper. Res.* **16** (1988) 201–240.

[3] J. Carlier and A.H.G. Rinnooy Kan, Scheduling subject to nonrenewable-resource constraints. *Oper. Res. Lett.* **1** (1982) 52–55.

[4] J. Carlier, A. Moukrim and H. Xu, The project scheduling problem with production and consumption of resources: a list-scheduling based algorithm. *Discrete Appl. Math.* **157** (2009) 3631–3642.

[5] J. Carlier, A. Moukrim and A. Sahli, Lower bounds for the event scheduling problem with consumption and production of resources. *Discrete Appl. Math.* **234** (2016) 178–194.

[6] J. Carlier, E. Pinson, A. Sahli and A. Jouglet, An $O(n^2)$ algorithm for time-bound adjustments for the cumulative scheduling problem. *Eur. J. Oper. Res.* **286** (2020) 468–476.

[7] J. Carlier, A. Sahli, A. Jouglet and E. Pinson, A faster checker of the energetic reasoning for the cumulative scheduling problem. *Int. J. Prod. Res.* (2021) 1–16. DOI: 10.1080/00207543.2021.1923853

[8] A. Cesta, A. Oddi and S. Smith, A constraint-based method for project scheduling with time windows. *J. Heuristics* **8** (2002) 109–136.

[9] S.J. Edwards, D. Baatar, K. Smith-Miles and A.T. Ernst, Symmetry breaking of identical projects in the high-multiplicity rcpsp/max. *J. Oper. Res. Soc.* **72** (2021) 1822–1843.
[10] S. Hartmann and D. Briskorn, An updated survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **297** (2021) 1–14.
[11] S. Johnson, Optimal two and three-stage production schedules with setup times included. *Nav. Res. Logistics Q.* **1** (1954) 61–68.
[12] E. Kaplan and A. Amir, A fast feasibility test for relocation problems. *Eur. J. Oper. Res.* **35** (1988) 201–205.
[13] I. Krimi, R. Benmansour, S. Hanafi and N. Elhachemi, Two-machine flow shop with synchronized periodic maintenance. *RAIRO-Oper. Res.* **53** (2019) 351–365.
[14] P. Laborie, Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artif. Intell.* **143** (2002) 151–188.
[15] L.V.D. Melo and T.A.D. Queiroz, Integer linear programming formulations for the RCPSP considering multi-skill, multi-mode, and minimum and maximum time lags. *IEEE Lat. Am. Trans.* **19** (2021) 5–16.
[16] C.L. Monma, The two-machine maximum flow time problem with series-parallel precedence constraints: an algorithm and extensions. *Oper. Res.* **27** (1979) 792–798.
[17] C.L. Monma and J.B. Sidney, Sequencing with series-parallel precedence constraints. *Math. Oper. Res.* **4** (1979) 215–224.
[18] K. Neumann and C. Schwindt, Project scheduling with inventory constraints. *Math. Methods Oper. Res.* **56** (2002) 513–533.
[19] K. Neumann and J. Zhan, Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *J. Intell. Manuf.* **19** (1997) 205–217.
[20] K. Neumann, C. Schwindt and J. Zimmermann, Resource-constrained project scheduling with time windows: recent developments and new applications. In: Perspectives in Modern Project Scheduling, edited by J. Jozefowska and J. Weglarz. Kluwer, Boston (2006) 375–407.
[21] K.V. Palem and B.B. Simons, Scheduling time-critical instructions on risc machines. *ACM Trans. Program. Lang. Syst.* **15** (1993) 632–658.
[22] C.H. Papadimitriou and M. Yannakakis, Scheduling interval-ordered tasks. *SIAM J. Comput.* **8** (1979) 405–409.
[23] Y. Sekiguchi, A decomposition theory based on a dominance relation and composite jobs. *Discrete Appl. Math.* **17** (1987) 187–211.
[24] R. Sethi, Complete register allocation problems. *SIAM J. Comput.* **4** (1975) 226–248.
[25] H. Singh, J.S. Oberoi and D. Singh, Multi-objective permutation and non-permutation flow shop scheduling problems with no-wait: a systematic literature review. *RAIRO-Oper. Res.* **55** (2021) 27–50.
[26] F. Sourd and J. Rogerie, Continuous filling and emptying of storage systems in constraint-based scheduling. *Eur. J. Oper. Res.* **165** (2005) 510–524.
[27] J. Valdes, R. Tarjan and E. Lawler, The recognition of series-parallel digraphs. *SIAM J. Comput.* **11** (1982) 298–313.