Published paper

Shakhlevich, N.V. (2008) *Preemptive scheduling on uniform parallel machines with controllable job processing times.* Algorithmica, 51 (4). pp. 451-473.
http://dx.doi.org/10.1007/s00453-007-9091-9

Algorithmica
DOI 10.1007/s00453-007-9091-9

Preemptive Scheduling on Uniform Parallel Machines with Controllable Job Processing Times

Natalia V. Shakhlevich · Vitaly A. Strusevich

Received: 28 February 2007 / Accepted: 8 March 2007

Abstract In this paper, we provide a unified approach to solving preemptive scheduling problems with uniform parallel machines and controllable processing times. We demonstrate that a single criterion problem of minimizing total compression cost subject to the constraint that all due dates should be met can be formulated in terms of maximizing a linear function over a generalized polymatroid. This justifies applicability of the greedy approach and allows us to develop fast algorithms for solving the problem with arbitrary release and due dates as well as its special case with zero release dates and a common due date. For the bicriteria counterpart of the latter problem we develop an efficient algorithm that constructs the trade-off curve for minimizing the compression cost and the makespan.

Keywords Uniform parallel machine scheduling · Controllable processing times · Generalized polymatroid · Maximum flow

1 Introduction

We study the scheduling model in which the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on uniform parallel machines $M_1, M_2, \ldots, M_m$. For each job, its processing time is not given in advance but has to be chosen by the decision-maker from a given range.

N.V. Shakhlevich (✉)
School of Computing, University of Leeds, Leeds LS2 9JT, UK
e-mail: ns@comp.leeds.ac.uk

V.A. Strusevich
Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, UK
e-mail: V.Strusevich@greenwich.ac.uk

For each job $j \in N$ we are given the interval $[\underline{p}_j, \overline{p}_j]$ from which the actual value p_j of the processing time is to be chosen. That selection process can be viewed either as *compressing* or *crashing* $\overline{p}_j$ down to p_j, or *decompressing* $\underline{p}_j$ up to p_j. In the former case, the value $x_j = \overline{p}_j - p_j$ is called the *compression amount* of job j, while in the latter case $z_j = p_j - \underline{p}_j$ is called the *decompression amount* of job j. Compression may decrease job completion time(s) but incurs additional cost $\alpha_j x_j$, where α_j is a given unit compression cost. The total cost associated with a choice of the actual processing times is represented by the linear function $\sum_{j \in N} \alpha_j x_j$.

The processing machines are *uniform,* i.e., machine M_h has speed s_h, $1 \le h \le m$. Without loss of generality, assume that the machines are numbered in non-increasing order of their speeds, i.e.,

$$s_1 \ge s_2 \ge \cdots \ge s_m. \tag{1}$$

A usual scheduling requirement says that a machine cannot process more than one job at a time, and a job is never processed on more than one machine at a time. Given a set of actual processing times p_j, the jobs can be processed with *preemption*. For some schedule, denote the total time during which a job $j \in N$ is processed on machine M_h, $1 \le h \le m$, by $q_j^{(h)}$. Taking into account the speed of the machine, we call the quantity $s_h q_j^{(h)}$ the *processing amount* of job j on machine M_h. It follows that

$$p_j = \sum_{h=1}^{m} s_h q_j^{(h)}.$$

Each job $j \in N$ is given a release date r_j, before which it is not available, and a due date d_j, by which it is desirable to complete its processing. Given a schedule, let C_j denote the completion time of job j, i.e., the time at which the last portion of job j is finished on the corresponding machine. A schedule is called *feasible* with respect to the due dates if $C_j \le d_j$ for all jobs $j = 1, \ldots, n$. The value $C_{\max} = \max_{j \in N} C_j$ determines the maximum completion time of all jobs and is called the *makespan*. We exclude from further consideration the case that $n < m$, since to minimize the makespan we need to process n jobs on n fastest machines.

The problem of our primal concern is to determine the values of actual processing times and find the corresponding schedule on m uniform machines so that all jobs meet their due dates and total compression cost is minimized. Extending standard notation for scheduling problems [11], we denote problems of this type by $Q|r_j, p_j = \overline{p}_j - x_j, C_j \le d_j, pmtn|\sum \alpha_j x_j$. Here, r_j in the middle field implies that the jobs have individual release dates; this parameter is omitted if the release dates are equal. We write $p_j = \overline{p}_j - x_j$ to indicate that the processing times are controllable and x_j is the compression amount to be found. The condition $C_j \le d_j$ reflects the fact that in a feasible schedule the due dates should be respected; we write $C_j \le d$ if the due dates are equal. The abbreviation $pmtn$ is used to point out that preemption is allowed. Finally, in the third field we write the objective function to be minimized.

In this paper we also consider a bicriteria problem $Q|p_j = \overline{p}_j - x_j, pmtn|(C_{\max}, K)$, where K denotes a compression cost function, namely $\sum \alpha_j x_j$. To describe a solution of a bicriteria problem we find the set of Pareto optimal points

Table 1 Time complexity of the algorithms with fixed processing times

Release times	Due dates	Objective	Running time	Reference
Arbitrary	Arbitrary	$C_j \le d_j$	$O(mn^3)$	[7]
Arbitrary	$d_j = d$	$C_j \le d$ (or $C_{\max}$)	$O(n \log n + nm)$	[10, 18]
$r_j = 0$	$d_j = d$	$C_j \le d$ (or $C_{\max}$)	$O(n + m \log m)$	[8]

defined by the break-points of the so-called efficiency frontier; see [23] for definitions and a state-of-the-art survey of multicriteria scheduling. Recall that a schedule S' is called Pareto optimal if there exists no schedule S'' such that $C_{\max}(S'') \le C_{\max}(S')$ and $K(S'') \le K(S')$, where at least one of these relations holds as a strict inequality.

Scheduling problems with controllable processing times have received considerable attention, see, e.g., surveys [12, 16]. The study of these problems is motivated by their numerous applications to various areas. Below we discuss only computing-related applications and refer to [20] for applications of scheduling with controllable processing times to manufacturing and operations management.

In real-time systems, a common goal is to schedule processors so that all computation is completed by specified deadlines because missing a deadline causes a system failure. However, meeting all timing requirements may not be possible for heavily loaded systems. In this case, some computations are performed partially, producing less precise results. For example, in computing systems that support imprecise computations, an iterative program or an image processing algorithm can often be logically decomposed. In our notation, a task with processing requirement $\overline{p}_j$ can be split into a mandatory part which takes $\underline{p}_j$ time, and an optional part that may take up to $\overline{p}_j - \underline{p}_j$ time. To produce a result of acceptable quality, the mandatory part must be completed in full, while an optional part improves the accuracy of the solution. If instead of an ideal computation time $\overline{p}_j$ a task is executed for $p_j = \overline{p}_j - x_j$ time, then computation is imprecise and x_j corresponds to the error of computation. In this application, total compression cost $\sum \alpha_j x_j$ is the total weighted error. See [12, 13, 21, 22].

A similar situation occurs in computer systems that collect data from sensing devices where the jobs can be completed partially in order to meet their deadlines, while incomplete jobs result in information loss (see [4], Sect. 5.4.2). Decision-making in this setting is based on finding the trade-off curve between the time required to complete all jobs and the value of lost information.

If the processing times are fixed, the known results on finding a due date feasible preemptive schedule and/or minimizing the makespan on m uniform parallel machines are given in Table 1.

If the processing times are controllable, problem $Q|r_j, \overline{p}_j - x_j, C_j \le d_j, pmtn|\sum \alpha_j x_j$ as a model of imprecise computation is studied by Błażewicz and Finke [3] and Leung [12] and shown to be solvable in $O(m^2n^4 \log mn + m^2n^3 \log^2 mn)$ and $O(m^2n^4 \log mn)$ time, respectively.

Nowicki and Zdrzalka [17] study problem $Q|\overline{p}_j - x_j, C_j \le d, pmtn|\sum \alpha_j x_j$ with equal release dates and a common due date. They claim that it is solvable in

$O(n \log n + nm)$ time, but it is not clear whether their algorithm can be extended to solve the bicriteria problem in polynomial time. They also give a pseudopolynomial-time approximation scheme for finding the efficient frontier for the bicriteria problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, K)$.

If the machines are identical, i.e., have equal speeds, then the following two single criterion problems are studied for controllable processing times: $P|\overline{p}_j - x_j, C_j \leq d, pmtn|\sum \alpha_j x_j$ and $P|r_j, \overline{p}_j - x_j, pmtn, C_j \leq r_j + d|\sum \alpha_j x_j$. The first problem reduces to the continuous linear knapsack problem [9, 20] and thus can be solved in $O(n)$ time while the second one reduces to linear programming [15]. The fastest algorithm for the bicriteria problem $P|\overline{p}_j - x_j, pmtn|(C_{\max}, K)$ requires $O(n \log n)$ time [20].

This paper is ideologically close to our previous work [20] in which several problems with controllable processing times have been reduced to optimizing a linear objective function over a special polyhedron, known as a polymatroid. In this paper, for more general scheduling problems we use a more general combinatorial structure of a generalized polymatroid, see Sect. 2 for relevant definitions. Here we only mention that the main advantage in using polymatroids is that we may simplify justification of the greedy approach to solving the corresponding scheduling problems, and eventually design simpler and faster algorithms than those known earlier.

The remainder of the paper is organized as follows. Section 2 reduces problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn|\sum \alpha_j x_j$ to maximizing a linear function over a generalized polymatroid which allows us to solve the problem in $O(mn^4)$ time by network flow techniques, faster than the earlier approaches [3, 12]. Section 3 addresses problem $Q|\overline{p}_j - x_j, C_j \leq d, pmtn|\sum \alpha_j x_j$ with equal release dates and a common due date. Using a polymatroidal representation we arrive at an $O(n \log n + nm)$-time algorithm. Compared with the algorithm by Nowicki and Zdrzalka, the justification, description and analysis of our algorithm is much easier and more natural than that in [17]. The bicriteria problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, K)$ is studied in Sect. 4. Again, the polymatroidal approach leads to an efficient and natural way of generating the break-points of the efficient frontier, thereby yielding the first polynomial-time algorithm for the bicriteria problem.

2 Arbitrary Release Times and Due Dates: Generalized Polymatroid

Consider the single criterion problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn|\sum \alpha_j x_j$ in which the jobs have arbitrary release times and due dates. Suppose that there are $q \leq 2n$ distinct values

$$t_1 < t_2 < \cdots < t_q$$

among r_j and d_j, $j = 1, \ldots, n$. These values divide the time line into $q - 1$ intervals $I_i = [t_i, t_{i+1}]$, $i = 1, \ldots, q - 1$. Denote the length of interval I_i by $\Delta_i := t_{i+1} - t_i$. Job j is *available* in interval I_i if $r_j \leq t_i$ and $d_j \geq t_{i+1}$. Within each interval I_i the set of available jobs does not change.

Define

$$S_h = \sum_{f=1}^{h} s_f,$$

the sum of the speeds of h fastest machines, $1 \le h \le m$. Taking into consideration the speed of each machine, notice that in an interval I_i total processing that could be done on machine M_1 is $s_1\Delta_i$, on machine M_2 is $s_2\Delta_i$, and so on.

Given actual values p_j of the processing times of the jobs, consider the problem of finding a feasible schedule in which each job j completes by its due date d_j. For a subset $A \subseteq N$ of jobs and an arbitrary interval I_i, let $A_i \subseteq A$ be the set of all jobs in A that are available in interval I_i. If $|A_i| \ge m$ then all m machines can be used for processing of the jobs of A_i in interval I_i; otherwise, the maximum total processing of the jobs of A_i in interval I_i will be achieved if $|A_i|$ fastest machines are used. Thus, for set A the maximum processing that can be performed in an interval I_i can be written as

$$\varphi_i(A) = \Delta_i S_{h_i}, \tag{2}$$

where $h_i = \min\{m, |A_i|\}$.

Let 2^N be a set of all subsets of the jobs from N. As proved by Martel (see Theorem 2.4.2 in [14]), a feasible schedule with job processing times p_j, $j = 1, \ldots, n$, exists if and only if the following conditions hold for each subset $A \in 2^N$ of jobs:

$$\sum_{j \in A} p_j \le \varphi(A),$$

where

$$\varphi(A) = \begin{cases} 0, & \text{if } A = \emptyset, \\ \sum_{i=1}^{q-1} \varphi_i(A), & \text{otherwise.} \end{cases} \tag{3}$$

If job processing times are controllable, then the values p_j are not known in advance, but are the decision variables. It is clear that the larger actual values p_j are, the smaller the total compression cost $\sum \alpha_j x_j$ is. Therefore, problem $Q|r_j, \overline{p}_j - x_j, C_j \le d_j, pmtn| \sum \alpha_j x_j$ reduces to the following linear program:

$$\max \sum \alpha_j p_j \tag{4}$$

subject to the constraints

$$\begin{aligned} &\sum_{j \in A} p_j \le \varphi(A), \quad \text{for all } A \in 2^N, \\ &\underline{p}_j \le p_j \le \overline{p}_j, \quad 1 \le j \le n. \end{aligned} \tag{5}$$

We show that the problem of maximizing the function (4) over the set of constraints (5) can be solved by a greedy algorithm by establishing the fact that (5) is a so-called generalized polymatroid. Recall several relevant definitions.

Definition 1 A set-function $\varphi : 2^N \to \mathbb{R}$ is called *submodular* if the inequality

$$\varphi(A \cup B) + \varphi(A \cap B) \leq \varphi(A) + \varphi(B)$$

holds for all A, B from 2^N.

It is also known (see, e.g., [19], p. 767) that $\varphi(A)$ is submodular if and only if the condition

$$\varphi(A \cup \{j,k\}) - \varphi(A \cup \{k\}) \leq \varphi(A \cup \{j\}) - \varphi(A) \tag{6}$$

holds for each $A \subseteq N$ and distinct j and k from $N \setminus A$.

Lemma 1 *Set-function $\varphi_i(A)$ of the form (2) is submodular.*

Proof We use the equivalent characterization of a submodular function (6). Recall that $A_i \subseteq A$ is the subset of jobs from A that are available in interval I_i, so that $\varphi_i(A) = \varphi_i(A_i)$.

If job j is not available in interval I_i, then

$$\begin{aligned}\varphi_i(A \cup \{j,k\}) - \varphi_i(A \cup \{k\}) &= \varphi_i(A_i \cup \{j,k\}) - \varphi_i(A_i \cup \{k\}) \\ &= \varphi_i(A_i \cup \{k\}) - \varphi_i(A_i \cup \{k\}) = 0,\\ \varphi_i(A \cup \{j\}) - \varphi_i(A) &= \varphi_i(A_i \cup \{j\}) - \varphi_i(A_i) = \varphi_i(A_i) - \varphi_i(A_i) = 0,\end{aligned}$$

so that (6) holds.

Similarly, if job k is not available in interval I_i, then we derive

$$\varphi_i(A \cup \{j,k\}) - \varphi_i(A \cup \{k\}) = \varphi_i(A_i \cup \{j\}) - \varphi_i(A_i) = \varphi_i(A \cup \{j\}) - \varphi_i(A),$$

as required.

Thus, in the remainder of this proof both jobs j and k are available in I_i. Denote $a = |A_i|$.

If $a \leq m-2$, then

$$\begin{aligned}\varphi_i(A_i \cup \{j,k\}) - \varphi_i(A_i \cup \{k\}) &= \Delta_i S_{a+2} - \Delta_i S_{a+1} = \Delta_i s_{a+2} \leq \Delta_i s_{a+1} \\ &= \Delta_i S_{a+1} - \Delta_i S_a = \varphi_i(A_i \cup \{j\}) - \varphi_i(A_i).\end{aligned}$$

If $a = m-1$, then

$$\begin{aligned}\varphi_i(A_i \cup \{j,k\}) - \varphi_i(A_i \cup \{k\}) &= \Delta_i S_m - \Delta_i S_m = 0 \leq \Delta_i s_m \\ &= \Delta_i S_m - \Delta_i S_{m-1} = \varphi_i(A_i \cup \{j\}) - \varphi_i(A_i).\end{aligned}$$

Finally, if $a \geq m$ then

$$\begin{aligned}\varphi_i(A_i \cup \{j,k\}) - \varphi_i(A_i \cup \{k\}) &= \Delta_i S_m - \Delta_i S_m = 0 \\ &= \Delta_i S_m - \Delta_i S_m = \varphi_i(A_i \cup \{j\}) - \varphi_i(A_i).\end{aligned}$$

Thus, (6) always holds. □

It is easy to check that each set-function φ_i of the form (2) is *monotone increasing*, i.e., $\varphi_i(A) \geq \varphi_i(B)$ for all sets $A \supseteq B$. It follows that function $\varphi(A)$ of the form (3) is submodular increasing as the sum of submodular increasing functions.

Definition 2 (Frank and Tardos [6], p. 495) A polyhedron

$$P_\varphi = \left\{ \mathbf{p} = (p_1, p_2, \ldots, p_n),\ \mathbf{p} \geq 0,\ \sum_{j \in A} p_j \leq \varphi(A) \text{ for each } A \in 2^N \right\}$$

is called a *polymatroid associated with* φ if function $\varphi(A)$ is a submodular, non-negative, monotone increasing and finite.

Thus, since function φ of the form (3) only takes non-negative finite values, it follows that P_φ is a polymatroid.

Here we refrain from giving the definition of the generalized polymatroid (or g-polymatroid); the reader is referred to [6], p. 501, or to [19], p. 845. For our purposes it suffices to mention that:

- a polymatroid is a g-polymatroid, see [19], p. 845;
- an intersection of a g-polymatroid with a box $B = \{\mathbf{p} \in \mathbb{R}^n, \underline{\mathbf{p}} \leq \mathbf{p} \leq \overline{\mathbf{p}}\}$ is a g-polymatroid, see [6], p. 507, and [19], p. 845;
- maximizing a linear function over a g-polymatroid can be done by a greedy algorithm, see [6], p. 524.

Thus, the constraints (5) define a g-polymatroid, and the problem of maximizing function (4) over it can be solved by the following algorithm.

Algorithm GrA

Step 1. If necessary, renumber the jobs so that

$$\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n. \tag{7}$$

Step 2. Define $p_k := \underline{p}_k$, $k = 1, \ldots, n$.
Step 3. FOR $k = 1$ to n do
Step 4. For job k, choose the decompression amount z_k as large as possible so that $(p_1, \ldots, p_n) \in P_\varphi$ for $p_k = \underline{p}_k + z_k$.
END FOR

For problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn| \sum \alpha_j x_j$ Algorithm GrA can be implemented in the following way. Starting with fully compressed processing times $p_j = \underline{p}_j$, $j = 1, \ldots, n$, the first task is to find an amount of processing of each job j in each interval I_i in a schedule with no late jobs (we assume that such a schedule exists, otherwise the problem makes no sense). Then for each job we need to find the largest decompression amount by considering the jobs one by one in accordance with numbering (7) as described in Step 4.

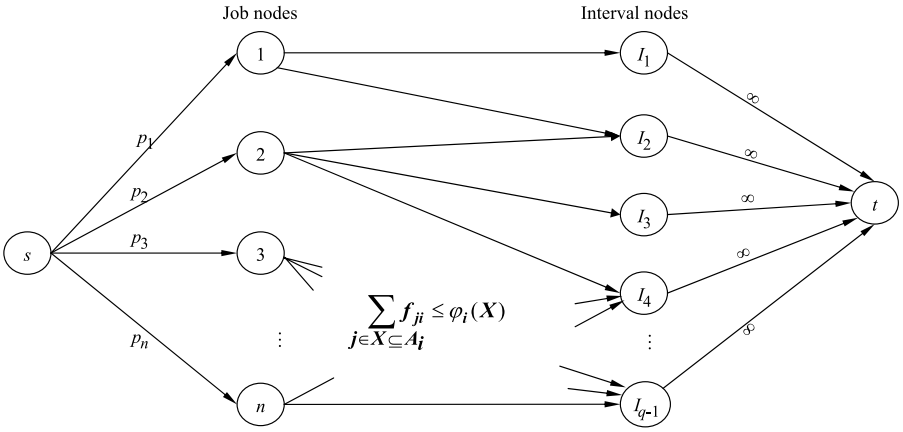

Fig. 1 Polymatroidal network by Martel

There are several ways of performing the first task. We can find the maximum polymatroidal flow of the total value $\sum_{j \in N} \underline{p}_j$ in the corresponding Martel's network (see Fig. 1); the running time of Martel's algorithm is $O(m^2 n^4 + n^5)$ [14]. There is a more computationally efficient way of performing this task which relies on an $O(mn^3)$-time algorithm by Federgruen and Groenvelt [7] of solving the (ordinary) maximum flow problem in a special network (see Fig. 2).

Martel's network contains n job nodes connected with the source by the arcs of capacities p_j, $j = 1, \ldots, n$, and $q - 1$ interval nodes connected with the sink by the arcs of infinite capacities. The job nodes are connected directly to the interval nodes: if job j is available in interval I_i, then the corresponding job node and the interval node are connected by an arc with capacity $S_1 \Delta_i$. Unlike the traditional maximum flow models, in the polymatroidal network additional constraints are introduced to define upper bounds on the cumulative capacities of the sets of arcs: if X_i is a (sub)set of arcs entering interval node I_i, then the total flow on the arcs of set X_i cannot be larger than $\varphi_i(X_i)$, where φ_i is defined by (2). If f_{ji} is a flow on the arc from job node j to interval node I_i, then the value f_{ji} can be viewed as the total amount of processing of job j in interval I_i in a feasible schedule.

Compared with Martel's network, in the Federgruen-Groenvelt network additional intermediate machine-interval nodes are introduced in-between the job nodes and the interval nodes. If job j can be processed in interval I_i, then job node j is connected with m machine-interval nodes $(I_i, 1), (I_i, 2), \ldots, (I_i, m)$ by the arcs with capacities $(s_1 - s_2)\Delta_i, (s_2 - s_3)\Delta_i, \ldots, (s_m - 0)\Delta_i$, respectively. These machine-interval nodes in their turn are connected to the interval node I_i by the arcs with capacities $1 \cdot (s_1 - s_2)\Delta_i, 2 \cdot (s_2 - s_3)\Delta_i, \ldots, m \cdot (s_m - 0)\Delta_i$. Similar to Martel's network, the total flow f_{ji} in the Federgruen-Groenvelt network from a job node j to an interval I_i defines the amount of processing of job j in interval I_i.

To implement Step 4 of Algorithm GrA any of the two network flow models can be used, Martel's or the one by Federgruen and Groenvelt. We describe a generic approach that does not depend on the model used.

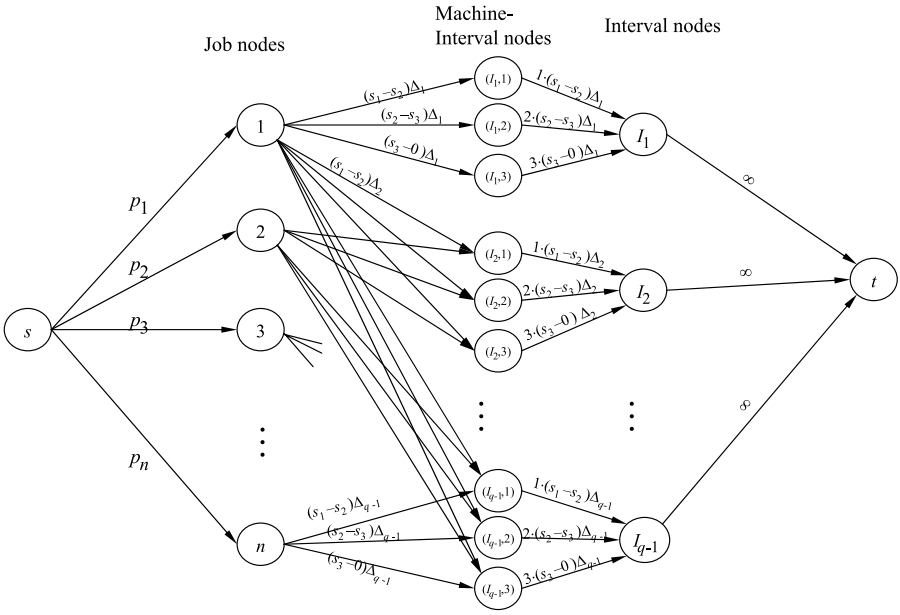

Fig. 2 Network flow model by Federgruen & Groenvelt

First, update the network by introducing for each arc from the source to each job node j a lower and upper bounds on the arc capacity, both equal to $\underline{p}_j$, $j \in N$. In a typical iteration (see Step 4) take the next job k and make an upper bound on the capacity of the arc entering job node k equal to $\overline{p}_k$. Find the maximum flow in the obtained network. Let p_k be the flow value on that arc. For further iterations, set both lower and upper bounds on the arc capacity to p_k. Accept p_k as the actual processing time of job k. Take the next unconsidered job with the largest compression cost, and so on.

In our implementation of Step 4 we select the Federgruen-Groenvelt model since for the networks we consider here, finding an ordinary maximum flow can be done in $O(mn^3)$ time [2, 7], faster than finding a polymatroidal maximum flow.

Recall that the process of finding the (ordinary) maximum flow with lower bounds consists of two stages: (i) finding a feasible flow and (ii) optimizing the flow; see [1]. In our case, the flow found in one iteration is feasible for the problem to be solved in the next iteration. The optimization stage deals with the so-called residual arc capacities and in our case still requires $O(mn^3)$ time.

Since Step 4 is repeated n times, we deduce that the overall running time of our implementation of Algorithm GrA applied to problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn|\sum \alpha_j x_j$ does not exceed $O\left(mn^4\right)$.

Recall that problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn|\sum \alpha_j x_j$ (or rather a relevant problem of minimizing the total late work) was studied by Błażewicz and Finke [3] and Leung [12] who suggested a reduction to the problem of finding the minimum cost flow in a modified Federgruen-Groenvelt network. Their approaches lead to the algorithms with the running times of $O(m^2n^4 \log mn + m^2n^3 \log^2 mn)$ and

$O(m^2n^4 \log mn)$, respectively, and are explicitly presented only for the case that $\underline{p}_j = 0$, $j \in N$.

3 Equal Release Times and Due Dates: Single Criterion

In this section we address the simplest version of problem $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn| \sum \alpha_j x_j$ in which all jobs are simultaneously available at time zero, i.e., $r_j = 0$, and they have to be completed by a common due date d. We give an efficient implementation of Algorithm GrA that solves this single criterion problem in $O(n \log n + nm)$ time.

We adapt conditions (5) from Sect. 2. Comparing problems $Q|r_j, \overline{p}_j - x_j, C_j \leq d_j, pmtn| \sum \alpha_j x_j$ and $Q|\overline{p}_j - x_j, C_j \leq d, pmtn| \sum \alpha_j x_j$, notice that in the latter problem all scheduling is done in a single interval $I_1 = [0, d]$. Thus, we deduce that problem $Q|\overline{p}_j - x_j, C_j \leq d, pmtn| \sum \alpha_j x_j$ reduces to maximizing the linear function (4) over a g-polymatroid determined by the constraints (5) with set-function $\varphi(A)$ defined for any $A \in 2^N$ as

$$\varphi(A) = \begin{cases} 0, & \text{if } A = \emptyset, \\ dS_h, & \text{otherwise,} \end{cases}$$

where

$$h = \min\{m, |A|\}.$$

Consider Step 4 of Algorithm GrA that finds the maximum decompression amount for job k. A straightforward implementation of this step may require checking exponentially many inequalities from (5) that define the polymatroid. For a more efficient implementation of Step 4, determine a permutation $\lambda = (\lambda(1), \lambda(2), \ldots, \lambda(n))$ of jobs such that

$$p_{\lambda(1)} \geq p_{\lambda(2)} \geq \cdots \geq p_{\lambda(n)}, \tag{8}$$

breaking ties so that the jobs with equal current processing times are sequenced in non-increasing order of their compression costs. Define

$$\begin{aligned} P_h(\lambda) &= \sum_{i=1}^{h} p_{\lambda(i)}, \quad h = 1, \ldots, m-1; \\ P_n &= \sum_{j=1}^{n} p_j. \end{aligned} \tag{9}$$

It is known (see, e.g., [5]) that a schedule with the makespan d exists if and only if

(i) for each h, $1 \leq h \leq m-1$, h longest jobs can be processed on h fastest machines, and
(ii) all jobs can be completed on all machines by time d,

so that

$$\begin{aligned} &P_h(\lambda) \le dS_h, \quad h = 1, \ldots, m-1, \\ &P_n \le dS_m. \end{aligned} \tag{10}$$

Clearly, it is sufficient to consider the latter m inequalities together with n box-inequalities of the form

$$\underline{p}_j \le p_j \le \overline{p}_j, \quad j = 1, \ldots, n \tag{11}$$

when finding the maximum decompression of job k. In what follows, we describe an efficient implementation of Algorithm GrA based on $m + n$ constraints (10–11) instead of an exponential number of constraints (5).

We start with fully crashed jobs with $p_j = \underline{p}_j$, $j \in N$. Let π be the current permutation of jobs, and in the beginning of the algorithm $\pi = \lambda$, where λ is defined by (8). During the process of decompression, the jobs will change their relative order with respect to the current processing times. Due to (9) it suffices to keep track of the first $m - 1$ positions of the current permutation π of jobs. Throughout the process, we maintain the following structure of that permutation.

Definition 3 Given the current values p_j of the processing times, permutation π of jobs is called the *main* permutation if

(i) the first $m - 1$ positions of π are occupied by the longest jobs sorted in non-increasing order of p_j; the jobs with equal processing times are additionally sorted in non-increasing order of their compression costs α_j;
(ii) the remaining jobs are placed starting from position m, the decompressible jobs (with $p_j < \overline{p}_j$) being positioned in non-increasing order of their compression costs α_j and followed by the fully uncrashed jobs (with $p_j = \overline{p}_j$) taken in any order.

Determine the slacks of the constraints (10) by

$$\tau_h = \begin{cases} dS_h - P_h(\pi), & \text{if } 1 \le h \le m-1, \\ dS_m - P_n, & \text{if } h = m. \end{cases}$$

Consider Step 4 of Algorithm GrA in which job $k = \pi(u)$ is subject to decompression. Observe that the position of job k in that permutation cannot be larger than m:

$$u \le m, \tag{12}$$

since any decompressible job in position $m + 1$ or larger has a smaller compression cost than that of job $\pi(m)$.

The current processing time p_k can be enlarged by z_k until one of the following events occurs:

Event A: job k becomes fully uncrashed, i.e., $p_k + z_k = \overline{p}_k$;
Event B: for some h, $1 \le h \le m$, slack τ_h becomes equal to zero;
Event C: the current value $p_k + z_k = p_{\pi(u)} + z_{\pi(u)}$ becomes equal to $p_{\pi(u-1)}$ for $u \le m$.

If Event A takes place and $u = m$, then in order to maintain the properties of π as the main permutation job k is moved to the last position; otherwise, if $u < m$, job k remains the u-th largest job and retains its position in the main permutation.

If Event B happens for $h = m$, then in the corresponding schedule all machines are permanently busy in the time interval $[0, d]$, so that no further decompression is possible without violating the deadline d.

If Event B takes place for $h \leq m - 1$, then in the corresponding schedule h longest jobs and only those are processed by h fastest machines, so that no further decompression of jobs $\pi(1), \ldots, \pi(h)$ is possible without violating the deadline d.

Notice that decompression of job $k = \pi(u)$, $u \leq m$, does not affect the values $P_h(\pi)$ and the corresponding slacks τ_h for $h \leq u - 1$. Therefore, the largest possible decompression z_k such that either Event A or Event B occurs is given by

$$z_k = \min\left\{\overline{p}_k - p_k, \tau_{\min}\right\},$$

where $\tau_{\min} = \min_{u \leq h \leq m}\{\tau_h\}$.

In the case of Event C, the processing time of job k takes the value $p_{\pi(u-1)}$ and that job still requires further decompressing in later iterations. In order to maintain the main permutation π while job k is being decompressed, this job is swapped with job $\pi(u-1)$. If there are more than one job with the processing time $p_{\pi(u-1)}$, job k is successively swapped with the jobs in earlier positions until it appears in front of all the jobs of this length.

We now give implementation details of Step 4 of Algorithm GrA for finding the optimal decompression amount z_k of job k in position $\pi(u)$ in the current main permutation π. It is assumed that we enter this step having found the values P_n, τ_m, and $P_h(\pi)$, τ_h for $h = 1, \ldots, m-1$.

Implementation of Step 4 of Algorithm GrA
(decompression of job $k = \pi(u)$, $u \leq m$)

Find $\tau_{\min} = \min_{u \leq h \leq m}\{\tau_h\}$.
WHILE $p_k < \overline{p}_k$ and $\tau_{\min} > 0$ DO
  Compute $z_k = \min\left\{\overline{p}_k - p_k,\ \tau_{\min},\ p_{\pi(u-1)} - p_k\right\}$.
  Increase p_k by z_k.
  Case A: $p_k = \overline{p}_k$. If $u = m$, move job k to position $\pi(n)$; otherwise no further actions required.
  Case B: $z_k = \tau_{\min}$. Set $\tau_{\min} := 0$; no further actions required.

Case C: $p_k = p_{\pi(u-1)}$. $j = 1$, $\tau_{\min} := \tau_{\min} - z_k$
WHILE $p_k = p_{\pi(u-j)}$ DO
swap job $k = \pi(u)$ with
job $\pi(u-j)$
$\tau_{\min} := \min\{\tau_{u-j}, \tau_{\min}\}$
$j := j + 1$
END WHILE
$u := u - j + 1$
END WHILE
Update the values $P_n, \tau_m, P_h(\pi)$ and τ_h for $h = 1, \ldots, m-1$ taking into account total decompression of job k.

Initial sorting of the machines in accordance with (1) and the jobs in accordance with (7) and (8) requires $O(n \log n)$ time. For the sorted jobs and machines, all sums $P_n, \tau_m, P_h(\pi)$ and τ_h, $1 \le h \le m-1$, can be calculated in $O(n)$ time.

In Step 4, for each job each of Events A or B may happen at most once. Due to (12), Event C occurs no more than m times. Thus, the total running time of Algorithm GrA is $O(n \log n + nm)$. Finding the corresponding optimal schedule requires $O(n + m \log m)$ time, see [8].

Example Consider the following instance of problem $Q|\overline{p}_j - x_j, C_j \le d, pmtn|\sum \alpha_j x_j$. There are $m = 3$ machines with the speeds $s_1 = 5$, $s_2 = 3$, $s_3 = 1$. The due date is $d = 10$. The processing times of $n = 4$ jobs and their compression costs are given in the table:

j	$\underline{p}_j$	$\overline{p}_j$	α_j
1	2	30	8
2	10	10	1
3	50	50	1
4	1	1	1

Observe that the jobs are numbered so that (7) holds.

In accordance with Algorithm GrA, we start with fully compressed jobs with $p_j = \underline{p}_j$, $1 \le j \le 4$, and the main permutation $\pi = (3, 2, 1, 4)$. We enter Step 4 of Algorithm GrA having found

$$P_1(\pi) = 50;\ P_2(\pi) = 50 + 10 = 60;\ P_4 = 50 + 10 + 2 + 1 = 63;$$
$$\tau_1 = 5 \times 10 - 50 = 0;$$
$$\tau_2 = (5 + 3) \times 10 - (50 + 10) = 20;$$
$$\tau_3 = (5 + 3 + 1) \times 10 - (50 + 10 + 2 + 1) = 27.$$

The first job that is subject to decompression is job $k = 1 = \pi(3)$ and $\tau_{\min} = \min_{3 \le h \le 3}\{\tau_h\} = \tau_3$. It follows that this job can be decompressed by $z_1 = \min\{30 - 2,\ 27,\ 10 - 2\} = 8$, i.e., Event C occurs. After that decompression p_1

becomes equal to 10 and we swap jobs 2 and 1 in permutation π so that π becomes $(3, 1, 2, 4)$ and $\tau_{\min}$ becomes $\min\{20,\ 19\} = 19$.

Job 1 is further decompressed by $z_1 = \min\{30 - 10,\ 19,\ 50 - 10\} = 19$, i.e., Event B occurs. As a result p_1 becomes equal 29 and $\tau_{\min} = 0$. No further decompression of any job is possible.

To finish Step 4, we update the values of $P_h(\pi)$ and τ_h:

$$P_1(\pi) = 50;\ P_2(\pi) = 50 + 29 = 79;\ P_4 = 50 + 29 + 10 + 1 = 90;$$
$$\tau_1 = 0;$$
$$\tau_2 = 1;$$
$$\tau_3 = 0.$$

4 Equal Release Times and Due Dates: Two Criteria

In this section, we develop a polynomial-time algorithm for the bicriteria problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, \sum \alpha_j x_j)$. As a solution of problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, \sum \alpha_j x_j)$, we find a sequence of all break-points of the efficient frontier $(C^0, K^0), (C^1, K^1), \ldots, (C^\gamma, K^\gamma), \ldots, (C^\Gamma, K^\Gamma)$, where C^γ is the makespan of the corresponding schedule and K^γ is the total compression cost, $0 \le \gamma \le \Gamma$. For each break-point (C^γ, K^γ) an actual schedule σ^γ can be found in $O(n + m \log m)$ time, see [8].

In order to construct the first break-point (C^0, K^0) we crash all jobs to their minimum processing times, i.e., set $p_j = \underline{p}_j$ for all $j,\ 1 \le j \le n$. Determine the main permutation π. Recall that in π the first $m - 1$ positions are occupied by the longest jobs taken in non-increasing order of their processing times. It is known (see, e.g., [5]) that for the fixed processing times p_j, $j \in N$, the optimum value of the makespan C is given by

$$C = \max\left\{P_n/S_m,\ \max_{1 \le h < m}\{P_h(\pi)/S_h\}\right\},$$

where $P_h(\pi)$ and P_n are defined by (9). Define $C^0 = C$. In general, in the corresponding schedule some jobs can be uncrashed without increasing C^0. The optimal processing times p_j, $\underline{p}_j \le p_j \le \overline{p}_j$, that minimize the total compression cost $\sum_{j=1}^n \alpha_j x_j$, can be found in $O(n \log n + nm)$ time by solving problem $Q|\overline{p}_j - x_j, pmtn, C_j \le C^0| \sum \alpha_j x_j$ with a common due date $d = C^0$ by Algorithm GrA. Denote the found schedule by σ^0. The corresponding value of K^0 is given by $\sum_{j=1}^n \alpha_j(\overline{p}_j - p_j)$, where p_j, $1 \le j \le n$, are actual processing times of the jobs found by Algorithm GrA.

Notice that a possible structure of schedule σ associated with a break-point (C, K) is such that

(I) either all machines are busy in the time interval $[0, C]$

or

(II) h fastest machines, where $h < m$, process h longest jobs in the time interval $[0, C]$, while the remaining slower machines process only fully uncrashed jobs; otherwise, the total cost can be decreased by increasing the processing time of a certain decompressible job without exceeding C.

Consider an arbitrary Pareto optimal schedule σ with the makespan C, cost K and the main permutation π. For this schedule the inequalities

$$\begin{aligned} &P_h(\pi) \le CS_h, \quad h = 1, \ldots, m-1, \\ &P_n \le CS_m, \end{aligned} \tag{13}$$

hold, at least one of them being the equality.

In schedule σ, take a decompressible job k located in the u-th position of π and define a *block* $B(u; 1, g)$ as a partial schedule for fully occupied machines $M_1, \ldots, M_g$, where $g \ge u$, such that in (13) the inequality for $h = g$ holds as equality and all inequalities for $h \in \{u, \ldots, g-1\}$, if any, are strict. In other words, the g-th inequality of (13) is the first inequality no higher than the u-th inequality that holds as equality.

Let $B(u; 1, g)$ be a block in schedule σ. If $g < m$, then in block $B(u; 1, g)$ the jobs of the set $N(u; 1, g) = \{\pi(1), \ldots, \pi(u), \ldots, \pi(g)\}$ are processed on the machines $M_1, \ldots, M_g$ and all these machines complete simultaneously at time C. On the other hand, if $g = m$ then in block $B(u; 1, g)$ the jobs of set $N(u; 1, g) = \{\pi(1), \ldots, \pi(u), \ldots, \pi(n)\} = N$ are processed on $M_1, \ldots, M_m$.

In line with the greedy argument, a transition from one Pareto optimal schedule to another is done by decompressing a (partially) crashed job $k = \pi(u)$ with the largest compression cost. Suppose that k belongs to some block $B(u; 1, g)$. As the processing time of job $k \in N(u; 1, g)$ grows, the structure of the corresponding schedule may change, and the situation that such a change takes place determines the next breakpoint of the efficient frontier. A possible structural change is associated with one of the following three events:

Event U: job k becomes fully uncrashed;
Event V: the processing time of job $k = \pi(u)$ becomes equal to that of the nearest longest job $\pi(u-1)$, so that the main permutation needs to be updated;
Event W: for block $B(u; 1, g)$ a new block $B(u; 1, g')$ emerges, where $u \le g' < g$.

For block $B(u; 1, g)$ denote

$$P = \begin{cases} P_n, & \text{if } g = m, \\ P_g(\pi), & \text{if } g < m. \end{cases}$$

Let z be a decompression amount of job k that is being decompressed. As the processing time p_k grows by z, the makespan of the resulting schedule increases by some value $\delta(z)$. The purpose of the next lemma is to establish the relationship between these two values.

Lemma 2 *Suppose that for schedule σ job $k = \pi(u)$ is subject to decompression and k belongs to block $B(u; 1, g)$. If job k is decompressed by amount $z_k > 0$ so that the earliest of three events U, V or W occurs, then in the resulting schedule machines*

$M_1, \ldots, M_g$ are permanently busy processing the jobs of set $N(u; 1, g)$ in the time interval $[0, C + \delta(z_k)]$, where

$$\delta(z_k) = \frac{z_k}{S_g}. \tag{14}$$

Proof Since $B(u; 1, g)$ is a block, then in the original schedule only the jobs of set $N(u; 1, g)$ are processed on machines $M_1, \ldots, M_g$ so that

$$P = CS_g.$$

As the processing time of job k grows by $z = z_k$ until the earliest of three events U, V or W occurs, still in the resulting schedule only the jobs of set $N(u; 1, g)$ are processed on machines $M_1, \ldots, M_g$:

$$P + z_k = (C + \delta(z_k))\, S_g,$$

i.e., the extra processing time z_k should be redistributed over the same set of machines. It follows from the above two equalities, that the makespan increases by $\delta(z_k) = z_k/S_g$. □

It is straightforward to verify that the decompression amount

$$z_k^U = \overline{p}_k - p_k$$

leads to Event U, while the decompression amount

$$z_k^V = p_{\pi(u-1)} - p_k$$

leads to Event V. Below we present the lemma that gives a formula for z_k^W leading to Event W as the earliest event.

Lemma 3 *Suppose that for schedule σ job $k = \pi(u)$ is subject to decompression and k belongs to block $B(u; 1, g)$. If Event W is the earliest event that occurs as a result of decompressing k by z_k^W, then*

$$z_k^W = \min_{u \le h \le g-1} \left\{ \frac{PS_h - P_h(\pi) S_g}{S_g - S_h} \right\}. \tag{15}$$

Proof Consider decompression of job k by an amount z. In accordance with (13) we have that

$$\begin{array}{ll} P_h(\pi) \le (C + \delta(z)) S_h, & h = 1, \ldots, u-1, \\ P_h(\pi) + z \le (C + \delta(z))\, S_h, & h = u, \ldots, g-1, \end{array} \tag{16}$$

and additionally,

$$P + z = (C + \delta(z))\, S_g.$$

Comparing these relations with (13), notice that for each of the first $u - 1$ inequalities only the right-hand side grows, so that all of them are strict inequalities. In order

to find out when the earliest Event W occurs we need to find the smallest increment $z = z_k^W$ such that one of the inequalities with $h = g'$, $u \le g' \le g-1$ in (16) becomes equality, i.e., block $B(u; 1, g')$ emerges.

For this value of z we have that

$$P_{g'}(\pi) + z_k^W = \big(C + \delta(z_k^W)\big) S_{g'}.$$

Since $\delta(z_k^W) = z_k^W / S_g$ due to Lemma 2, it follows that

$$z_k^W = \frac{S_g(C S_{g'} - P_{g'}(\pi))}{S_g - S_{g'}} = \frac{P S_{g'} - P_{g'}(\pi) S_g}{S_g - S_{g'}} > 0.$$

Thus, the value of z_k^W can be found by taking the minimum of the values $\frac{P S_h - P_h(\pi) S_g}{S_g - S_h}$ over all h, $u \le h \le g-1$, which corresponds to formula (15). □

Suppose that for the original problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, \sum \alpha_j x_j)$ we have found a sequence of break-points $(C^0, K^0), \ldots, (C^{\gamma-1}, K^{\gamma-1})$. At point $(C^{\gamma-1}, K^{\gamma-1})$ we know the actual processing times equal to $p_j = \underline{p}_j + z_j$, the main permutation π and the corresponding schedule $\sigma^{\gamma-1}$ that satisfies the conditions where at least one inequality (13) with $C = C^{\gamma-1}$ holds as the equality. We describe a transition to the next break-point (C^{γ}, K^{γ}).

For schedule $\sigma^{\gamma-1}$ determine a decompressible job k with the largest cost α_k. Assume that job k is located in the u-th position of π. We start with considering the case that schedule $\sigma^{\gamma-1}$ consists of a single block $B(u; 1, m)$ that includes all machines.

The decompression amount for job $k = \pi(u)$ that leads to the earliest of the Events U, V or W is equal to

$$\begin{aligned} z_k &= \min\{z_k^U,\ z_k^V,\ z_k^W\} \\ &= \min\left\{\overline{p}_k - p_k,\ \ p_{\pi(u-1)} - p_k,\ \min_{u \le h \le m-1}\left\{\frac{P_n S_h - P_h(\pi) S_m}{S_m - S_h}\right\}\right\}, \end{aligned}$$

where the last right-hand side term follows from (15).

If job k is decompressed by z_k, we obtain the next schedule σ^{γ} that corresponds to the break-point (C^{γ}, K^{γ}) of the efficient frontier, where

$$\begin{aligned} C^{\gamma} &= C^{\gamma-1} + \frac{z_k}{S_m}; \\ K^{\gamma} &= K^{\gamma-1} - \alpha_k z_k. \end{aligned}$$

Consider now the situation that in schedule $\sigma^{\gamma-1}$ job $k_1 = \pi(u_1)$ to be decompressed belongs to block $B(u_1; 1, g_1)$, $g_1 < m$. For further purposes we refer to this block as $B(u_1; g_0 + 1, g_1)$, where for completeness g_0 is set equal to 0. Similar to the previous case we can find the decompression amount $z_{k_1} = \min\{z_{k_1}^U,\ z_{k_1}^V,\ z_{k_1}^W\}$ of job k_1 that leads to the earliest of Events U, V or W, and determine the corresponding

increase $\delta_1 = z_{k_1}/(S_{g_1} - S_{g_0})$ in the makespan for the jobs of set $N(u_1; g_0 + 1, g_1)$ processed in block $B(u_1; g_0 + 1, g_1)$; here for completeness $S_{g_0} = 0$.

If this decompression were performed as described, we would not obtain a Pareto optimal schedule, since further decompression is possible for one or several jobs of the set $\{\pi(g_1 + 1), \ldots, \pi(n)\}$. Thus, we look for a decompressible job $k_2 = \pi(u_2)$ such that $u_2 > g_1$ and the compression cost α_{k_2} is the largest among all decompressible jobs of set $N \backslash N(u_1; g_0 + 1, g_1)$. Temporarily disregard block $N(u_1; g_0 + 1, g_1)$, and consider the subproblem of a smaller size for processing the jobs of set $N \backslash N(u_1; g_0 + 1, g_1)$ on machines $M_{g_1+1}, \ldots, M_m$. Similar to the above, for job u_2 find block $B(u_2; g_1 + 1, g_2)$, where g_2 is the smallest index, $g_2 \geq u_2$, for which in (13) with $C = C^{\gamma-1}$ the corresponding inequality holds as equality. If $g_2 < m$, then in this block the jobs of the set $N(u_2; g_1 + 1, g_2) = \{\pi(g_1 + 1), \ldots, \pi(u_2), \ldots, \pi(g_2)\}$ are processed on the machines $M_{g_1+1}, \ldots, M_{g_2}$ and all these machines complete simultaneously at time $C^{\gamma-1}$. On the other hand, if $g_2 = m$ then in block $B(u_2; g_1 + 1, g_2)$ the jobs of set $N(u_2; g_1 + 1, g_2) = \{\pi(g_1 + 1), \ldots, \pi(u_2), \ldots, \pi(n)\}$ are processed on $M_{g_1+1}, \ldots, M_m$. Considering block $B(u_2; g_1 + 1, g_2)$ in a similar way as block $B(u_1; g_0 + 1, g_1)$ we can derive the formulas for the largest possible decompression amount z_{k_2} of job $k_2 = \pi(u_2)$ that leads to the earliest Event U, V or W in block $B(u_2; g_1 + 1, g_2)$, and the corresponding increment $\delta_2 = z_{k_2}/(S_{g_2} - S_{g_1})$ of the makespan of the jobs of set $N(u_2; g_1 + 1, g_2)$. Again, if $g_2 = m$, there are no further actions to be taken; otherwise, we search for the next decompressible job, the corresponding block, etc.

This process of identifying the decompressible jobs continues until we find a job $k_y = \pi(u_y)$ to be decompressed simultaneously with all previously found jobs $k_\ell = \pi(u_\ell)$, $\ell = 1, \ldots, y - 1$. The corresponding value g_y for block $B(u_y; g_{y-1} + 1, g_y)$ is either equal to m or less than m. In the former case, the structure of schedule $\sigma^{\gamma-1}$ is described by property (I) so that all jobs of set $N(u_y; g_{y-1} + 1, g_y) = \{\pi(g_{y-1} + 1), \ldots, \pi(u_y), \ldots, \pi(n)\}$ are processed on machines $M_{g_{y-1}+1}, \ldots, M_m$, while in the latter case, the structure of schedule $\sigma^{\gamma-1}$ is described by property (II) so that the jobs of set $N(u_y; g_{y-1} + 1, g_y) = \{\pi(g_{y-1} + 1), \ldots, \pi(u_y), \ldots, \pi(g_y)\}$ and only those are processed on machines $M_{g_{y-1}+1}, \ldots, M_{g_y}$, and all jobs that follow job $\pi(g_y)$ in the main permutation π are fully uncrashed.

Thus, we have decomposed the problem of finding the next breakpoint into y subproblems Q_ℓ, $\ell = 1, \ldots, y$. In each subproblem Q_ℓ we consider jobs of set $N(u_\ell; g_{\ell-1} + 1, g_\ell)$ on machines $M_{g_{\ell-1}+1}, \ldots, M_{g_\ell}$. For each block $B(u_\ell; g_{\ell-1} + 1, g_\ell)$, $1 \leq \ell \leq y$, the largest decompression amount z_{k_ℓ} of job $k_\ell = \pi(u_\ell)$ that leads to the earliest Event U, V or W in block B_{g_ℓ} is equal to

$$
\begin{aligned}
z_{k_\ell} &= \min\{z^U_{k_\ell},\ z^V_{k_\ell},\ z^W_{k_\ell}\} \qquad (17)\\
&= \min\left\{\overline{p}_{k_\ell} - p_{k_\ell},\quad p_{\pi(u_\ell-1)} - p_{k_\ell},\ \min_{u_\ell \leq h \leq g_\ell - 1}\left\{\frac{PS_h - P_h(\pi)S_{g_\ell}}{S_{g_\ell} - S_h}\right\}\right\},
\end{aligned}
$$

where

$$
P = \begin{cases} P_n & \text{for } g_\ell = m,\\ P_{g_\ell}(\pi) & \text{for } g_\ell \leq m - 1. \end{cases}
$$

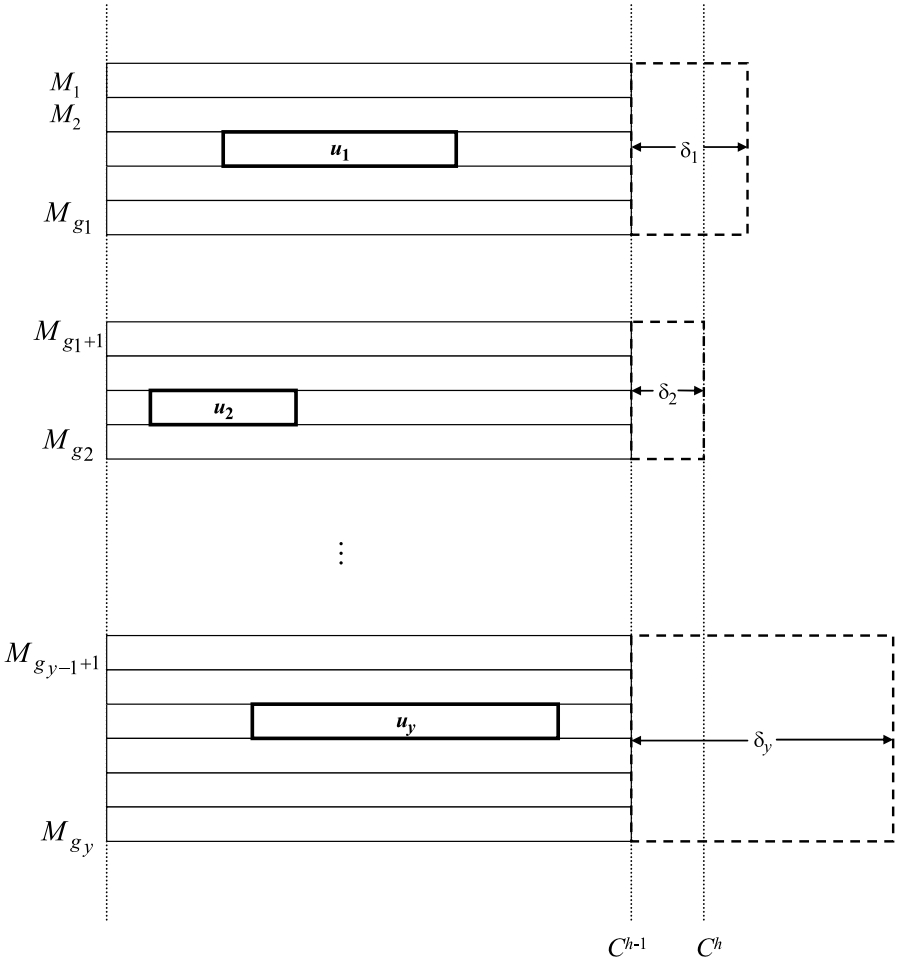

Fig. 3 Decompressing jobs $N_{g_1}, N_{g_2}, \ldots, N_{g_y}$

Additionally, define

$$\delta_\ell = \frac{z_{k_\ell}}{S_{g_\ell} - S_{g_{\ell-1}}}, \tag{18}$$

see Fig. 3.

To guarantee that the structure of schedule σ^γ that defines the next break-point (C^γ, K^γ) satisfies one of the properties (I) or (II), we need to find the largest possible value of δ, so that $C^\gamma = C^{\gamma-1} + \delta$ and the earliest event U, V or W occurs in one of the blocks:

$$\delta = \min\{\delta_\ell | 1 \le \ell \le y\}. \tag{19}$$

Having found increment δ in the makespan value, the processing time of each job $k_\ell = \pi(u_\ell)$ increases by $z_{k_\ell}^{\gamma-1}$ (see Lemma 2):

$$z_{k_\ell}^{\gamma-1} = \delta\left(S_{g_\ell} - S_{g_{\ell-1}}\right), \quad 1 \le \ell \le y. \tag{20}$$

Observe that the values $z_{k_\ell}^{\gamma-1}$ are defined in such a way that no Event U, V or W occurs if the processing times of jobs k_ℓ are decompressed by less than $z_{k_\ell}^{\gamma-1}$, $\ell = 1, \ldots, y$.

Summarizing, we can formalize the transition from break-point $(C^{\gamma-1}, K^{\gamma-1})$ to (C^γ, K^γ) as follows.

Algorithm *Transition* $(C^{\gamma-1}, K^{\gamma-1}) \to (C^\gamma, K^\gamma)$

Given: schedule $\sigma^{\gamma-1}$ with makespan $C^{\gamma-1}$ and compression cost $K^{\gamma-1}$, actual job processing times p_j, $j \in N$, and the main permutation π.

Initialization: set $\delta := \infty$, $g_0 := 0$, $\ell := 0$, $N' := N$.

1. WHILE $g_\ell < m$ and set N' contains jobs that are not fully uncrashed
   - 1.1 $\ell := \ell + 1$;
   - 1.2 Find a decompressible job k_ℓ such that $\alpha_{k_\ell} = \max_{j \in N'}\left\{\alpha_j | p_j < \overline{p}_j\right\}$. Let $k_\ell = \pi(u_\ell)$. Find block $B(u_\ell; g_{\ell-1}+1, g_\ell)$.
   - 1.3 Compute z_{k_ℓ} and δ_ℓ by formulas (17) and (18), respectively.
   - 1.4 Update $N' := N' \setminus \{\pi(g_{\ell-1}+1), \ldots, \pi(g_\ell)\}$ and set $y := \ell$.

   END WHILE
2. Compute δ by formula (19) and determine the decompression amounts $z_{k_\ell}^{\gamma-1}$ in accordance with (20). Decompress jobs $k_1, k_2, \ldots, k_y$ by the amounts $z_{k_1}^{\gamma-1}, z_{k_2}^{\gamma-1}, \ldots, z_{k_y}^{\gamma-1}$, respectively.
3. Set $C^\gamma := C^{\gamma-1} + \delta$, $K^\gamma := K^{\gamma-1} - \sum_{\ell=1}^{y} \alpha_{k_\ell} z_{k_\ell}^{\gamma-1}$ and update the main permutation π.

Let us estimate the running time of a single transition $(C^{\gamma-1}, K^{\gamma-1}) \to (C^\gamma, K^\gamma)$. Since each job k_ℓ is sought for among the first m elements of the main permutation π and their total number y does not exceed m, all these jobs together with the jobs g_ℓ will be found in $O(m^2)$ time. Computing z_{k_ℓ} requires at most $O(g_\ell - g_{\ell-1})$ time due to (17). Since $\sum_{\ell=1}^{y}(g_\ell - g_{\ell-1}) \le m$, it follows that Steps 1 and 2 together can be implemented in $O(m^2)$ time.

In Step 3, permutation π should be updated if either Event U or Event V occurs for job k_ℓ. In case of Event U, the job either remains in its current position or is moved to the last position, which can be implemented in constant time. In case of Event V, job

Table 2 Time complexity of the algorithms

Release times	Due dates	Objective	Earlier known results		This paper
Arbitrary	Arbitrary	$C_j \le d_j$	$O(m^2n^4 \log mn + m^2n^3 \log^2 mn)$	[3]	$O(mn^4)$
			$O(m^2n^4 \log mn)$ (min-cost-max-flow)	[12]	
$r_j = 0$	$d_j = d$	$C_j \le d;\ C_{\max}$	$O(n \log n + mn)$	[17]	$O(n \log n + mn)$

k_ℓ is swapped with all the preceding jobs that have the same processing time. Even if each job k_ℓ is moved to the first position of its block, this can be achieved for job k_ℓ in no more than m swaps. Thus, the overall time complexity of a single transition $(C^{\gamma-1}, K^{\gamma-1}) \to (C^\gamma, K^\gamma)$ is $O(m^2)$, provided that the starting permutation π and the partial sums P_g and S_g are known.

It follows that the running time needed for solving the original bicriteria problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, \sum \alpha_j x_j)$ is at most $O(n \log n + \Gamma m^2)$, where Γ is the number of break-points. To determine Γ we count the overall number of times that each Event U, V and W may take place.

Event U occurs no more than n times. Event V occurs no more than nm times, since each time a job to be swapped is located no further than in position m of the current permutation π. Finally, Event W may occur no more than m times in-between two consecutive Events U or V since each block $B(u_\ell; g_{\ell-1} + 1, g_\ell)$ can be split in at most $g_\ell - g_{\ell-1} \le m$ sub-blocks. Therefore, the number of break-points Γ does not exceed $O(nm^2)$.

This makes the overall time complexity of finding the efficient frontier equal to $O(n \log n + nm^4)$. Recall that for each Pareto-optimal point finding the corresponding optimal schedule requires $O(n + m \log m)$ time, see [8].

Observe that the approach developed by Nowicki and Zdrzalka in [17] allows finding an ε-approximation of the efficient frontier in pseudopolynomial $O(nm(\overline{C} - C^0)/\varepsilon)$ time, where $\overline{C}$ and C^0 are the optimal makespan values if all jobs are fully uncrashed and fully crashed, respectively.

5 Conclusions

In this paper, we have extended the polymatroidal approach suggested in [20] for preemptive single and identical parallel machine problems with controllable parameters to a more general scheduling model with uniform parallel machines. The main outcome is establishing the link between the most general type of preemptive scheduling problems with polymatroids. This allows us to provide a unified framework for solving the corresponding problems with controllable processing times.

The paper affirms that the polymatroidal approach simplifies justification of the greedy approach to solving the corresponding scheduling problems and eventually

leads to the design of simpler and faster algorithms than those known earlier. The results for single criterion problems are summarized in Table 2.

For the bicriteria problem $Q|\overline{p}_j - x_j, pmtn|(C_{\max}, \sum \alpha_j x_j)$ of minimizing makespan and compression cost we have developed an algorithm that required $O(n \log n + nm^4)$ time and constructs the breakpoints of the efficient frontier. It is the first algorithm that solves this bicriteria problem with uniform machines in polynomial time.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, New Jersey (1993)
2. Ahuja, R.K., Orlin, J.B., Stein, C., Tarjan, R.E.: Improved algorithms for bipartite network flow. SIAM J. Comput. **23**, 906–933 (1994)
3. Błażewicz, J., Finke, G.: Minimizing mean weighted execution time loss on identical and uniform processors. Inf. Process. Lett. **24**, 259–263 (1987)
4. Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling Computer and Manufacturing Processes. Springer, Berlin (2001)
5. Brucker, P.: Scheduling Algorithms, 4th edn. Springer, Berlin (2004)
6. Frank, A., Tardos, E.: Generalized polymatroids and submodular flows. Math. Program. **42**, 489–563 (1988)
7. Federgruen, A., Groenvelt, H.: Preemptive scheduling of uniform machines by ordinary network flow techniques. Manag. Sci. **32**, 341–349 (1986)
8. Gonzalez, T.F., Sahni, S.: Preemptive scheduling of uniform processor systems. J. ACM **25**, 92–101 (1978)
9. Jansen, K., Mastrolilli, M.: Approximation schemes for parallel machine scheduling problems with controllable processing times. Comput. Oper. Res. **31**, 1565–1581 (2004)
10. Labetoulle, J., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Preemptive scheduling of uniform machines subject to release dates. In: Pulleyblank, H.R. (ed.) Progress in Combinatorial Optimization, pp. 245–261. Academic, New York
11. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: algorithms and complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (eds.) Handbooks in Operations Research and Management Science, vol. 4, Logistics of Production and Inventory, pp. 445–522. North-Holland, Amsterdam (1993)
12. Leung, J.Y.-T.: Minimizing total weighted error for imprecise computation tasks. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling: Algorithms, Models and Performance Analysis. Computer and Information Science Series, pp. 34-1–34-16. Chapman & Hall/CRC, London (2004)
13. Leung, J.Y.-T., Yu, V.K.M., Wei, W.-D.: Minimizing the weighted number of tardy task units. Discrete Appl. Math. **51**, 307–316 (1994)
14. Martel, C.: Preemptive scheduling with release times, deadlines, and due dates. J. ACM **29**, 812–829 (1982)
15. Mastrolilli, M.: Notes on max flow time minimization with controllable processing times. Computing **71**, 375–386 (2003)
16. Nowicki, E., Zdrzalka, S.: A survey of results for sequencing problems with controllable processing times. Discrete Appl. Math. **26**, 271–287 (1990)
17. Nowicki, E., Zdrzalka, S.: A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times. Discrete Appl. Math. **63**, 271–287 (1995)
18. Sahni, S., Cho, Y.: Scheduling independent tasks with due times on a uniform processor system. J. ACM **27**, 550–563 (1980)
19. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, New York (2003)
20. Shakhlevich, N.V., Strusevich, V.A.: Preemptive scheduling problems with controllable processing times. J. Sched. **8**, 233–253 (2005)

21. Shih, W.-K., Liu, J.W.S., Chung, J.-Y.: Fast algorithms for scheduling imprecise computations. In: Proceedings of the 10th Real-time Systems Symposium, Santa-Monica, pp. 12–19 (1989)
22. Shih, W.-K., Liu, J.W.S., Chung, J.-Y.: Algorithms for scheduling imprecise computations with timing constraints. SIAM J. Comput. **20**, 537–552 (1991)
23. T'kindt, V., Billaut, J.-C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Berlin (2002)