Journal of Scheduling
https://doi.org/10.1007/s10951-023-00782-w

Preemptive scheduling of parallel jobs of two sizes with controllable processing times

Akiyoshi Shioura[1] · **Vitaly A. Strusevich**[2] · **Natalia V. Shakhlevich**[3]

Accepted: 21 February 2023

Abstract
In parallel machine scheduling, a size of a job is defined as the number of machines that are simultaneously required for its processing. This paper considers a scheduling problem in which the set of jobs consists of jobs of two sizes: the conventional jobs (each job requires a single machine for its processing) and parallel jobs (each job to be simultaneously processed on more than one machine). The processing times are controllable, and they have to be chosen from given intervals in order to guarantee the existence of a preemptive schedule in which all jobs are completed by a common deadline. The objective is to minimize the total compression cost which reflects possible reductions in processing times. Unlike problems of classical scheduling with conventional jobs, the model under consideration cannot be directly handled by submodular optimization methods. We reduce the problem to maximizing the total weighted work of all jobs parametrized with respect to total work of the parallel jobs. The solution is delivered by separately finding the breakpoints for the maximum total weighted work of the parallel jobs and for the maximum total weighted work of the conventional jobs. This results in a polynomial-time algorithm that is no slower than needed to perform sorting of jobs' parameters.

Keywords Multi-processor jobs · Submodular optimization · Parallel machine scheduling · Controllable processing times

1 Introduction

One of the main assumptions of the classical scheduling theory is that each job is not processed on more than one machine at a time. This assumption leads to a range of scheduling models that have found numerous applications in industry and services. However, during the last thirty years there has been a considerable volume of studies of various scheduling models in which a job can be (or even must be) processed on several machines simultaneously. The main application area of models of this type is the scheduling of computational tasks on computer systems of parallel architecture.

Formally, in scheduling on (identical) parallel machines, we are given a set $N = \{1, 2, \ldots, n\}$ of jobs/tasks to be processed on $m \geq 2$ machines/processors of set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. It takes $p(j)$ time units to process a job $j \in N$. Preemption is allowed, i.e., the processing of any job j can be interrupted and resumed later; the total length of time intervals of processing job j must be equal to $p(j)$. The following variants of scheduling models can be distinguished:

- **Classical:** each job can be processed preemptively on any machine, but only one at a time;
- **Dedicated:** for each job $j \in N$ a set of machines, traditionally denoted by $fixed(j) \subseteq \mathcal{M}$, is given and the job must be preemptively processed on all machines of this set simultaneously;
- **Rigid** (also known as non-moldable): for each job $j \in N$ a number of machines, traditionally denoted by $size(j)$, where $1 \leq size(j) \leq m$, is given and at any time of its processing the job must be preemptively processed on exactly $size\,(j)$ machines simultaneously;
- **Non-Rigid** (also known as malleable and moldable): for each job $j \in N$, its size is not fixed in advance, and a job is allowed to be preemptively processed on any number of machines, but its actual processing time depends on the number of these machines.

✉ Natalia V. Shakhlevich
n.shakhlevich@leeds.ac.uk

[1] Department of Industrial Engineering and Economics, Tokyo Institute of Technology, Tokyo 152-8550, Japan

[2] Welling, Kent, UK

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

Published online: 07 April 2023

By contrast with the classical model for which all jobs are *conventional*, i.e., of size one, in the last three models listed above some jobs require (or are allowed to use) more than one machine. We call such jobs *parallel* jobs or *multi-processor* jobs. See the monograph (Drozdowski, 2009) for a comprehensive exposition of scheduling with multi-processor jobs.

The main scheduling model studied in this paper belongs to the class of the rigid models. In fact, we focus on a restricted model, in which conventional jobs are processed by one machine and all parallel jobs have the same size Δ, where $1 < \Delta \leq m$.

Let S be a schedule that is feasible for a certain scheduling model. The completion time of job $j \in N$ is denoted by $C(j)$. Assuming that the processing times $p(j)$ are *fixed*, the following *feasibility* problems are of interest: to check whether there exists a schedule in which all jobs are completed by a given deadline D. Using the accepted scheduling notation, we denote these problems on m parallel identical machines by $P\,|pmtn, C(j) \leq D|-$ for the classical model and by $P\,|pmtn, size(j) \in \{1, \Delta\}, C(j) \leq D|-$ for the model with parallel jobs of two sizes. Each of these problems is known to be solvable in $O(n)$ time; see (McNaughton, 1959) for the classical model and Błażewicz et al. (1986) for the model with parallel jobs of two sizes. Notice that if the sizes of jobs are arbitrary, the problem is still solvable in $O(n)$ provided the number of machines m is fixed (Jansen & Porkolab, 2003).

In a more general setting, the processing times $p(j)$ are not fixed, but should be chosen by a decision-maker from a given interval $[l(j), u(j)]$. Selecting actual processing time $p(j)$ implies *compression* of the longest processing time $u(j)$ by the amount $x(j) = u(j) - p(j)$. Compression may decrease the completion time of each job j but incurs additional cost $w(j)x(j)$.

A wide range of the problems of *scheduling with controllable processing times* (SCPT) has been studied, see the most recent surveys (Shabtay & Steiner, 2007) and Shioura et al. (2018). For problems relevant to this paper, the goal is to minimize the total compression cost $\sum_{j\in N} w(j)x(j)$ for a schedule in which all jobs complete by a common deadline D. In particular, the problem on m identical parallel machines that we denote by $P\,|pmtn, p(j) = u(j) - x(j), C(j) \leq D|$ $\sum w(j)x(j)$ is solvable in $O(n)$ time (Jansen & Mastrolilli, 2004). The best known polynomial-time algorithms for more general problems, that allow job-dependent release dates and/or deadlines and machine speeds are developed in McCormick (1999); Shioura et al. (2013, 2015, 2016); see also the survey (Shioura et al., 2018). Methods for solving speed scaling scheduling problems for the energy-aware environment are presented in Shioura et al. (2017). All these algorithms are designed by adapting methods of submodular optimization.

All prior results on the problems with changeable processing times deal with the models of the classical type, i.e., those with the jobs of size one. The notable exception is scheduling multi-processor jobs on machines with changeable speeds; see (Kononov & Kovalenko, 2020a, b; Li, 1999, 2012).

Rigid tasks are typical for modern distributed systems, which allow users to formulate their specific resource requirements in terms of the CPU cores, GPU's, memory, input/output devices, etc. In a Gantt chart, rigid jobs are usually represented by two-dimensional rectangles, where $size(j)$ represents the requirement of job j in terms of resource utilization and $p(j)$ represents the job execution time. As stated in the survey (Lopes & Menasce, 2016), the study of rigid jobs is among most popular research directions.

Scenarios which combine the two features, task rigidness and controllability, arise, for example, in scientific computation. Users of scientific computation software, such as MAGMA (MAGMA, 2022), PETSc (PETSc, 2022), or ScaLAPACK (ScaLAPACK, 2022), are required to specify their resource requirement $size(j)$. A certain level of flexibility in computation time $p(j)$ is usually admitted. A solution of a higher precision is expected if computation time is $p(j) > l(j)$. The penalty for delivering a less accurate solution, compared to a solution found during the maximum computation time $u(j)$, is measured as $w(j)x(j)$. The associated scheduling problem combines then the features of scheduling with controllable processing times and rigid parallel tasks, with pre-specified resource requirements $size(j)$ for all jobs, $j \in N$.

Other scenarios of similar nature arise in the context of audio and video streaming, location analysis of moving objects, or temperature/humidity analysis of controlled environments. For these domains, getting a timely approximate result before the deadline is a preferred option compared to a delayed result of higher quality.

The ability to control processing times can be part of the service level agreement between a user and a resource provider. Such an agreement reflects an obligation of a resource provider to commit $size(j)$ resources during the time $u(j)$ for processing task j; it also regulates a penalty ($w(j)x(j)$ in our model) payable by the resource provider if job j gets required resources not for the time $u(j)$, but for a shorter time $p(j)$; the latter situation happens if the job is terminated earlier by the resource provider due to the adopted overload management policy.

The main objective of this paper is to verify to which extend the methods used for solving the SCPT problems with conventional jobs can be applied to handling the models which include the parallel jobs. In particular, we would like to see whether the techniques of submodular optimization can be used, directly or partly, for solving the SCPT problems with parallel jobs, even in the basic case of jobs of two sizes.

We present a polynomial-time algorithm for the problem with multi-processor jobs of two sizes that have to be completed before a common deadline and the total compression cost is to be minimized. Combining the earlier used pieces of notation, we denote this problem by $P\,|pmtn,\ p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \leq D| \sum w(j)x(j)$. Interestingly, the submodular optimization toolkit, elaborated in our earlier study, is applicable for the special cases of this problem, if either $size(j) = 1$ or $size(j) = \Delta$ for all jobs N, but not for the combined problem, with two types of jobs. As we show in this paper, the combined problem brings substantial algorithmic challenges; still the submodular optimization methods are helpful in the development of a fast algorithm. Our study can be considered as the first step toward developing the solution methods for the general problem, with jobs of different sizes.

In Sect. 2, we demonstrate that, unlike many SCPT problems of the classical setting, the problem with parallel jobs cannot be directly formulated as a linear programming problem with submodular constraints. An algorithm to handle the problem is outlined in Sect. 3. Sections 4 to 9 present detailed analysis of various parts of the algorithm. Further improvements for some components of the algorithm are presented in Appendix A. Section 10 contains concluding remarks.

2 Formulations and links to submodular optimization

For a problem with the jobs of two sizes, a job of a size 1 will be called a 1-job, while a job of size Δ will be called a Δ-job. Let N_1 and N_Δ denote the set of all 1-jobs and the set of all Δ-jobs, respectively, $N = N_1 \cup N_\Delta$.

Throughout this paper, for an n-dimensional vector $\mathbf{p} = (p(1), p(2), \ldots, p(n)) \in \mathbb{R}^N$ of processing times and a set $X \subseteq N$ define

$$p(X) = \sum_{j \in X} p(j).$$

We refer to this value as the *total work* of set X. Similarly, we define the minimum and the maximum work of set X:

$$l(X) = \sum_{j \in X} l(j),$$
$$u(X) = \sum_{j \in X} u(j).$$

In scheduling theory, the *processing capacity function* is a set-function $\varphi(X)$ defined for any subset of jobs $X \subseteq N$ and it represents the total capacity of the machines available for processing the jobs of set X. It is widely used, not only for the problems with fixed processing times, but also for the SCPT problems (Shioura et al., 2018).

In the classical setting, i.e., when each job is a 1-job, the function $\varphi(X)$ is essentially equal to the length of all time intervals within which the jobs of set X can be processed. This means that for a problem with fixed processing times a feasible schedule exists if and only if for each set X of jobs their total work does not exceed the available processing capacity. Thus, a feasible schedule exists if and only if the inequality

$$p(X) \leq \varphi(X) \tag{2.1}$$

holds for all sets $X \subseteq N$; see (Horn, 1974). For example, in the classical problem $P\,|pmtn, C(j) \leq D| -$ with m parallel identical machines and a common deadline D, each job of a set $X \subseteq N$ must be completed by time D, and we have that

$$\varphi(X) = \min\{|X|, m\} D.$$

More general feasibility problems on parallel machines, including the problems with job-dependent release dates and machine speeds can be found in Shioura et al. (2013, 2015); see also the survey (Shioura et al., 2018).

Let us try to derive the processing capacity function for the feasibility problem $P|pmtn,\ size(j) \in \{1, \Delta\}, C(j) \leq D| -$ and formulate the condition for the existence of a feasible schedule. For a set $X \subseteq N$ of jobs, let $X_1 = N_1 \cap X$ and $X_\Delta = N_\Delta \cap X$ be the sets of all 1-jobs and of all Δ-jobs, respectively, in X. Then the total processing requirement, i.e., the length of all intervals during which the jobs of set X have to be processed on the machines is given by $p(X_\Delta)\Delta + p(X_1)$. On the other hand, the total duration of the intervals that are available for this processing is given by

$$\varphi(X) = \min\{|X_\Delta|\,\Delta + |X_1|, m\} D.$$

However, a simple argument which defines the total processing requirement and the processing capacity in terms of lengths of the relevant intervals cannot be extended from a classical settings with 1-jobs only to the setting with parallel jobs. Indeed, for problem $P\,|pmtn, size(j) \in \{1, \Delta\}, C(j) \leq D| -$ the condition that the inequality

$$p(X_\Delta)\Delta + p(X_1) \leq \min\{|X_\Delta|\,\Delta + |X_1|, m\} D \tag{2.2}$$

holds for any set $X = X_1 \cup X_\Delta \subseteq N$ is a necessary condition for the existence of a feasible schedule, but not a sufficient condition.

To illustrate this, consider the following instance of $P\,|pmtn, size(j) \in \{1, \Delta\}, C(j) \leq D| -$ with $m = 3, n = 3$ jobs and $D = \Delta = 2$; each job is a 2-job of unit length, i.e., $N = N_\Delta = \{1, 2, 3\}$ and $p(j) = 1$ for $j \in N$. Clearly, for

this instance a feasible schedule in which all jobs complete by time $D = 2$ does not exist. But if we check the conditions (2.2) for a set $X \subseteq N = \{1, 2, 3\}$, we obtain the inequalities

$$2p(X) \le \min\{\Delta, m\}\ D = 4,\ \ |X| = 1;$$
$$2p(X) \le \min\{2\Delta, m\}\ D = 6,\ \ |X| = 2;$$
$$2p(X) \le \min\{3\Delta, m\}\ D = 6,\ \ |X| = 3,$$

which all hold, provided that $p(j) = 1,\ j \in N$.

Notice that for the instances with Δ-jobs only, we may assume that the number of machines is a multiple of Δ and any of the extra machines is empty in any feasible schedule and can be removed. Thus, if in the example above the third machine is removed, checking the conditions (2.2) does not lead to the contradiction.

One of the reasons for the successful development of fast algorithms for the SCPT problems on parallel machines in the classical settings is that for most environment the problem of minimizing total compression cost can be written as a linear programming (LP) problem over a region related to a submodular polyhedron.

For completeness, below we briefly discuss the relevant issues, some of which will be applied in the remainder of this paper. We mainly follow a comprehensive monograph on submodular optimization (Fujishige, 2005), see also (Katoh & Ibaraki, 1998) and Schrijver (2003).

Definition 1 (Fujishige, 2005) A set-function $\varphi : 2^N \to \mathbb{R}$ is called `submodular` if the inequality

$$\varphi(X) + \varphi(Y) \ge \varphi(X \cup Y) + \varphi(X \cap Y)$$

holds for all sets $X, Y \in 2^N$. For a submodular function φ defined on 2^N such that $\varphi(\emptyset) = 0$, the pair $(2^N, \varphi)$ is called a `submodular system` on N, while φ is referred to as its `rank function`.

Definition 2 (Fujishige, 2005) For a submodular system $(2^N, \varphi)$, polyhedra

$$P(\varphi) = \{\mathbf{p} \in \mathbb{R}^N \mid p(X) \le \varphi(X), \quad X \in 2^N\};$$
$$B(\varphi) = \{\mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P(\varphi), \quad p(N) = \varphi(N)\},$$

are called the `submodular polyhedron` and the `base polyhedron`, respectively, associated with the submodular system.

An SCPT problem with preemption can be modeled as the linear programming problem

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in N} w(j)p(j) && (2.3)\\ \text{subject to} \quad & p(X) \le \varphi(X), && X \in 2^N,\\ & l(j) \le p(j) \le u(j), && j \in N, \end{aligned}$$

where $\varphi(X)$ is the processing capacity function, $\mathbf{p} \in \mathbb{R}^N$ is a vector of decision variables, $\mathbf{w} \in \mathbb{R}^N_+$ is a non-negative weight vector, and $\mathbf{l}, \mathbf{u} \in \mathbb{R}^N$ are vectors of upper and lower bounds on the components of vector $\mathbf{p}$, respectively. This problem serves as a mathematical model for many SCPT problems to minimize total compression cost, including problem $P\,|pmtn,\ p(j) = u(j) - x(j), C(j) \le D|\sum w(j)x(j)$ and many of its extensions. This is due to the fact that the values of $p(j)$ are actual processing times in a feasible schedule and minimizing total compression cost

$$\sum_{j \in N} w(j)x(j) = \sum_{j \in N} w(j)(u(j) - p(j))$$

is equivalent to maximizing *total weighted processing time* or *total weighted work* $\sum_{j \in N} w(j)p(j)$.

If in problem (2.3) set-function $\varphi : 2^N \to \mathbb{R}$ is submodular and $\varphi(\emptyset) = 0$, then the feasible region of problem (2.3) is a *submodular polyhedron intersected with a box* (Shioura et al., 2018), and the problem can be solved by various tools of submodular optimization, including the classical greedy algorithm and a recent decomposition algorithm developed in Shioura et al. (2015).

Unfortunately, the submodular optimization toolkit available for solving the SCPT in the classical settings, with the jobs of size 1 only, cannot be used directly even for a relatively simple problem $P\,|pmtn,\ p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D|\sum_{j \in N} w(j)x(j)$. One reason for that is that there is no known reformulation of the corresponding feasibility problem in terms of capacity set-functions.

Suppose, however, such a formulation is known with respect to a capacity function φ, which is submodular. The problem can be written as an LP problem similar to (2.3); however, the feasible region will not be a submodular polyhedron, since the coefficients in the left-hand side of some constraints are different from 0 and 1.

For illustration, consider an instance of problem $P\,|pmtn,\ size(j) \in \{1, 2\}, C(j) \le D|-$ with $m = 2$ machines, $n = 3$ jobs and $\Delta = 2$ such that $N_\Delta = \{1\}$ and $N_1 = \{2, 3\}$. The capacity function φ is such that $\varphi(X) = 2D$ for $X = \{1\}$ and all sets X with $|X| \ge 2$, while $\varphi(\{j\}) = D$ for $j \in \{2, 3\}$. The set of constraints that describe a feasible region is written as

$$\begin{aligned} 2p(1) \ &\le 2D;\ p(2) \le D;\ p(3) \le D;\\ 2p(1) \ &+ p(2) \le 2D;\ 2p(1) + p(3) \le 2D;\\ p(2) \ &+ p(3) \le 2D;\\ 2p(1) \ &+ p(2) + p(3) \le 2D. \end{aligned}$$

which is not a submodular polyhedron.

Thus, in order to solve problem $P|pmtn,\ p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D|\sum_{j \in N} w(j)x(j)$ we

need to develop an alternative approach, which cannot be completely based on previously known techniques of submodular optimization.

3 Outline of our algorithm

Denote

$$n_\Delta = |N_\Delta|\,,\ n_1 = |N_1|\,.$$

Throughout this paper, we assume that the jobs are numbered in such a way that

$$N_\Delta = \{1, 2, \ldots, n_\Delta\}\,, \qquad N_1 = \{n_\Delta + 1, \ldots, n\}\,,$$

where we additionally assume that the jobs of set N_Δ are numbered in non-increasing order of their weights, i.e.,

$$w(1) \geq w(2) \geq \cdots \geq w(n_\Delta). \tag{3.1}$$

We also assume that a sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n_1))$ is a permutation of jobs in N_1 satisfying the condition

$$w(\sigma(1)) \geq w(\sigma(2)) \geq \cdots \geq w(\sigma(n_1)), \tag{3.2}$$

and the values

$$U_t = \sum_{j=1}^{t} u(\sigma(j)), \qquad L_t = \sum_{j=1}^{t} l(\sigma(j)),$$
$$t = 0, 1, \ldots, n_1, \tag{3.3}$$

are available. Notice that these assumptions can be satisfied in $O(n \log n)$ time.

We may assume that $u(j) \leq D$ for each $j \in N$; otherwise, the upper bounds can be reduced to D. Let $p(1), p(2), \ldots, p(n)$ be the variables that represent the values of the actual processing times such that $l(j) \leq p(j) \leq u(j)$ for each $j \in N$.

We introduce a parameter λ, equal to total work of the Δ-jobs, i.e.,

$$\lambda = \sum_{j=1}^{n_\Delta} p(j). \tag{3.4}$$

Provided that total work of Δ-jobs is equal to λ and the deadline D is observed, introduce the following functions of parameter λ:

- $W_\Delta(\lambda)$, the maximum total weighted work of Δ-jobs;
- $W_1(\lambda)$, the maximum total weighted work of 1-jobs;
- $W(\lambda)$, the maximum total weighted work of all jobs.

The structure of the problem allows us to employ the following approach to finding $W(\lambda)$. First, we consider the Δ-jobs alone and find their maximum total work $W_\Delta(\lambda)$; see Sect. 4. Then, keeping the structure of the corresponding schedule for processing the Δ-jobs, we add the 1-jobs and maximize $W_1(\lambda)$, see Sect. 5. As a result, $W(\lambda) = W_\Delta(\lambda) + W_1(\lambda)$. Note that the schedule for Δ-jobs is constructed in such a way that it does not create restrictions on finding the maximum total weighted work $W_1(\lambda)$ for 1-jobs, as it leaves room for scheduling 1-jobs in the most favorable way.

As mentioned in Sect. 2, the problem of minimizing total compression cost is equivalent to the problem of maximizing total weighted work, i.e., to maximizing $W(\lambda)$ over the appropriate range of λ-values. In the following sections, we describe an algorithm for finding $\lambda = \lambda^*$ that maximizes the function $W(\lambda)$; as soon as the value λ^* is obtained, we can also obtain the corresponding optimal processing times $p^*(j)$ of jobs $j \in N$.

Below we present an outline of the proposed algorithm. In Sects. 4, 5, and 6, we show that for a given λ, the function value $W(\lambda)$ can be computed in $O(n)$ time, provided that the sorted lists of the input parameters $w(j)$, $u(j)$, and $l(j)$ are given. We also show that each function $W_\Delta(\lambda)$, $W_1(\lambda)$, and $W(\lambda)$ is a piecewise-linear concave function. Therefore, an optimal value of λ that maximizes $W(\lambda)$ is achieved at some breakpoint of that function, and we describe how to search for such a breakpoint.

Any breakpoint of $W(\lambda)$ is either a breakpoint of $W_\Delta(\lambda)$ or a breakpoint of $W_1(\lambda)$. We classify all breakpoints of $W(\lambda)$ into two types: *major* breakpoints and *minor* breakpoints.

Major breakpoints consist of breakpoints of $W_\Delta(\lambda)$, plus some additional points (see Sect. 4 for details). We show in Sect. 4 that all major breakpoints can be obtained in $O(n_\Delta)$ time. Major breakpoints split the domain of the parameter λ into intervals, and we want to find an interval that contains λ^*, an optimal value of λ. We show in Sect. 7 that by using binary search, we can find in $O(n \log n_\Delta)$ time at most two candidates for such interval.

Minor breakpoints consist of breakpoints of function $W_1(\lambda)$. We show in Sect. 8 that for any single interval defined by the major breakpoints, the number of the minor breakpoints in the interval is $O(n_1)$. In Sect. 8, we present an $O((n_1)^2)$-time algorithm for finding all minor breakpoints. Then, we apply binary search to find a minor breakpoint that maximizes the function value $W(\lambda)$, which is a global maximizer of $W(\lambda)$, and this operation can be done in $O(n \log n_1)$ time.

In total, function $W(\lambda)$ can be maximized in $O(n \log n + (n_1)^2)$ time.

Theorem 1 *Problem* $P|pmtn, p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D| \sum_{j \in N} w(j)x(j)$ *can be solved in* $O(n \log n + (n_1)^2)$ *time.*

It is easy to see that the lower bound on the time complexity of the algorithm that solves this scheduling problem is $O(n \log n)$, and the bound $O((n_1)^2)$ for computing the minor breakpoints is the bottleneck to achieve the best possible time bound. In Appendix A, we present a faster algorithm for computing the minor breakpoints and show that it runs in $O(n_1 \log n_1)$ time. This makes it possible to solve our scheduling problem in $O(n \log n)$ time.

Theorem 2 *Problem* $P|pmtn, p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D| \sum_{j \in N} w(j)x(j)$ *can be solved in* $O(n \log n)$ *time.*

In the following sections, we explain each part of the outlined algorithm in detail.

4 Analysis of total weighted work of parallel jobs

In this section, we show that $W_\Delta(\lambda)$, an optimal total weighted work of the Δ-jobs, provided that their total work is equal to λ, is a piecewise-linear concave function with $O(n_\Delta)$ breakpoints, and it can be computed in $O(n_\Delta)$ time. It is assumed that all Δ-jobs are numbered in non-increasing order of their weights, in accordance with (3.1).

Notice that in a feasible schedule the total work λ of Δ-jobs is contained in the interval $[\underline{\lambda}, \overline{\lambda}]$, where

$$\underline{\lambda} = \sum_{j \in N_\Delta} l(j), \quad \overline{\lambda} = \min \left\{ \sum_{j \in N_\Delta} u(j), \ \lfloor m/\Delta \rfloor D \right\}. \quad (4.1)$$

For a fixed $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, the function value $W_\Delta(\lambda)$ is equal to the optimal value of the following linear programming problem:

$$\begin{aligned} &\text{Maximize} \sum_{j \in N_\Delta} w(j)p(j) \quad (4.2)\\ &\text{subject to} \sum_{j \in N_\Delta} p(j) = \lambda,\\ &\qquad l(j) \le p(j) \le u(j), \ j \in N_\Delta. \end{aligned}$$

The feasible region of problem (4.2) is a base polyhedron intersected with a box, and the problem can be solved by a greedy algorithm. The algorithm considers the jobs in the order of their numbering and each job j is assigned the largest value of $p(j)$ which does not affect the previous assignments.

For $j = 0, 1, \ldots, n_\Delta$, we define

$$\lambda_j = \sum_{j'=1}^{j} u(j') + \sum_{j'=j+1}^{n_\Delta} l(j').$$

Notice that

$$\lambda_0 = \sum_{j \in N_\Delta} l(j) = \underline{\lambda}, \quad \lambda_{n_\Delta} = \sum_{j \in N_\Delta} u(j),$$
$$\lambda_0 \le \lambda_1 \le \cdots \le \lambda_{n_\Delta}.$$

Provided that $\lambda \in [\lambda_{q-1}, \lambda_q]$, $1 \le q \le n_\Delta$, an optimal solution to problem (4.2), can be found by computing actual processing times $p^*(j)$, $j \in N_\Delta$, by

$$p^*(j) = \begin{cases} u(j) & \text{for } j = 1, 2, \ldots, q-1, \\ \lambda - \lambda_{q-1} + l(q) & \text{for } j = q, \\ l(j) & \text{for } j = q+1, q+2, \ldots, n_\Delta, \end{cases}$$

and the corresponding optimal value of the objective function is given by

$$W_\Delta(\lambda) = \sum_{i=1}^{q-1} w(i)u(i) + \sum_{i=q+1}^{n_\Delta} w(i)l(i) + w(q)(\lambda - \lambda_{q-1} + l(q)). \quad (4.3)$$

The following statement holds.

Lemma 1 *For any given* $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, *the function value* $W_\Delta(\lambda)$ *can be computed in* $O(n_\Delta)$ *time.*

It follows from formula (4.3) that function $W_\Delta(\lambda)$ is piecewise-linear and its slope is decreasing in λ as λ changes within $[\underline{\lambda}, \overline{\lambda}]$. Therefore, the following is valid.

Lemma 2 *Function* $W_\Delta(\lambda)$ *is a piecewise-linear concave function.*

We see that breakpoints of the function $W_\Delta(\lambda)$ are given by

$$\{\underline{\lambda}, \overline{\lambda}\} \cup \{\lambda_j \mid 0 \le j \le n_\Delta, \ \lambda_j \in [\underline{\lambda}, \overline{\lambda}]\}.$$

In addition to these points, we consider the set of points

$$\{rD \mid r \text{ is integer}, \underline{\lambda} \le rD \le \overline{\lambda}\}.$$

The points in the union of these two sets are called the *major breakpoints* of function $W(\lambda)$. In the following discussion, we often consider a closed interval $I = [\lambda', \lambda'']$ given by two consecutive major breakpoints λ' and λ''; we call such an interval a *major interval*.

By definition, major breakpoints consist of breakpoints of $W_\Delta(\lambda)$ and additional points rD; due to these additional points, any major interval I satisfies the condition that

$$I \subseteq [rD, (r+1)D] \quad \text{for some integer } r. \tag{4.4}$$

This means that if $\lambda \in [\lambda', \lambda'']$, where $[\lambda', \lambda'']$ is a major interval, then all Δ-jobs can be processed by using the same number of groups of Δ machines. This property makes it easier to analyze the function $W_1(\lambda)$ in Sect. 8.

It is easy to see that the number of breakpoints of the function $W_\Delta(\lambda)$ is at most $n_\Delta + 2$. We have

$$\overline{\lambda} \le \sum_{j=1}^{n_\Delta} u(j) \le Dn_\Delta$$

since by assumption $u(j) \le D$ holds for $j \in N_\Delta$. Hence, if $rD \le \overline{\lambda}$, then we have $r \le n_\Delta$, and this implies that

$$|\{rD \mid r \text{ is integer}, \underline{\lambda} \le rD \le \overline{\lambda}\}| \le n_\Delta.$$

Thus, the number of the major breakpoints is $O(n_\Delta)$.

The procedure below computes all major breakpoints of function $W(\lambda)$ and outputs them as an increasing list.

Procedure ComputeMajor

Step 1. Compute $\underline{\lambda}$ and $\overline{\lambda}$ by (4.1). Set $\lambda_0 := \underline{\lambda}$.
Step 2. For $j = 1, 2, \ldots, n_\Delta$, set $\lambda_j := \lambda_{j-1} + u(j) - l(j)$.
Step 3. Let j' (respectively, j'') be the smallest (respectively, the largest) integer j such that $\lambda_j \in (\underline{\lambda}, \overline{\lambda})$.
Step 4. Let r' be the smallest integer such that $r'D \ge \underline{\lambda}$. Let r'' be the largest integer such that $r''D \le \overline{\lambda}$.
Step 5. Merge the sorted lists $\underline{\lambda}, \lambda_{j'}, \lambda_{j'+1}, \ldots, \lambda_{j''}, \overline{\lambda}$ and $r'D, (r'+1)D, \ldots, r''D$, eliminate duplicates (if any), and output the merged sorted list.

It is easy to see that Procedure ComputeMajor requires $O(n_\Delta)$ time. Hence, we obtain the following lemma.

Lemma 3 *The number of major breakpoints of function $W(\lambda)$ is $O(n_\Delta)$, and all major breakpoints can be found in $O(n_\Delta)$ time.*

To illustrate various algorithmic aspects of the paper, we will use the instance of $P|pmtn,\ p(j) = u(j) - x(j),\ size(j) \in \{1, \Delta\},\ C(j) \le D| \sum_{j \in N} w(j)x(j)$ presented below.

Example 1. There are 6 jobs to be processed on 5 machines so that they all should be completed by time $D = 7$. The parameters of the jobs are given in Table 1. It is assumed that $\Delta = 2$ and each job of set $N_\Delta = \{J_1, J_2, J_3\}$ requires two machines for its processing, while each job of set $N_1 = \{J_4, J_5, J_6\}$ is conventional. Notice that the jobs of each of these two sets are numbered in decreasing order of their weights.

Table 1 Data for Example 1

j	$l(j)$	$u(j)$	$w(j)$
1	2	4	5
2	1	6	3
3	3	5	1
4	2	6	5
5	1	8	2
6	3	5	1

In accordance with Procedure ComputeMajor, we find $\underline{\lambda} = 6$ and $\overline{\lambda} = \min\{15, 14\} = 14$. We also compute $\lambda_0 = 6, \lambda_1 = 8, \lambda_2 = 13$ and $\lambda_3 = 15$. Determine $j' = 1$ and $j'' = 2$, as well as $r' = 1$ and $r'' = 2$. The list of the major breakpoints is given by $(6, 7, 8, 13, 14)$.

To illustrate (4.3), take, e.g., $\lambda \in [\lambda_1, \lambda_2]$, and compute $W_\Delta(\lambda) = w(1)u(1) + w(3)l(3) + w(2)\left(\lambda - \lambda_1 + l\left(2\right)\right) = 20 + 3 + 3(\lambda - 7) = 3\lambda + 2$. In particular for $\lambda = 10$, we have that $W_\Delta(\lambda) = 32$ and the corresponding optimal processing times of the Δ-jobs are $p^*(1) = 4$, $p^*(3) = 3$ and $p^*(2) = 3$; notice that the sum of these processing times is equal to $\lambda = 10$.

5 Analysis of total weighted work of 1-jobs

In this section, we show that $W_1(\lambda)$ is a piecewise-linear concave function.

5.1 Explicit formula for $W_1(\lambda)$

Let $\lambda \in \left[\underline{\lambda}, \overline{\lambda}\right]$ be a taken value of total work of the Δ-jobs. To determine the maximum total weighted work $W_1(\lambda)$ of the 1-jobs, we consider the case that $rD < \lambda \le (r+1)D$, where $r = \lfloor \lambda / D \rfloor$.

For λ in this range, the Δ-jobs are assigned to r groups of Δ machines, occupying these machines fully in $[0, D]$, and additionally the remaining load of the Δ-jobs equal to $\lambda - rD$ is assigned to another group of Δ machines. Time $D - (\lambda - rD) = (r+1)D - \lambda$ left on each of the machines of the latter group has to be used to process the 1-jobs. Note that leaving only one group of Δ machines partially loaded is the best possible configuration for scheduling 1-jobs. Thus, to determine $W_1(\lambda)$ we need to solve a problem with controllable processing times to maximize weighted processing time of 1-jobs, provided that Δ machines are only available in the interval $[0, d]$, where $d = (r+1)D - \lambda$, while the remaining $m' = m - (r+1)\Delta$ machines are available in the

interval $[0, D]$. We say that the Δ machines that are available in the interval $[0, d]$ are *restricted*, and the remaining m' machines are *unrestricted*.

Example 2. For illustration of what we want to achieve, take the same instance as in Example 1. Assume that $\lambda = 10$. The processing times of the size 2 jobs are as found in Example 1. We see that $r = 1$, i.e., we can build a schedule in which one group of two machines is busy in the interval $[0, 7]$ and one group is busy for three time units, e.g., in the interval $[4, 7]$. More specifically, machines M_1 and M_2 simultaneously process job J_1 in the time interval $[0, 4]$ and job J_2 in $[4, 7]$, while machines M_3 and M_4 are restricted and simultaneously process job J_3 in the interval $[4, 7]$.
Since $r = 1$ and $m' = 1$, we deduce that for the processing of the conventional jobs J_4, J_5 and J_6 we may use the restricted machines M_3 and M_4, each available in the interval $[0, d] = [0, 4]$, and the unrestricted machine M_5 available in the interval $[0, 7]$.
Our goal is to maximize the total weighted work $W_1(\lambda)$ of the conventional jobs. Further in this section we explain how to achieve this in general by formulating and solving the corresponding problem with submodular constraints. Before we do this, we use our small size example to illustrate the logic behind such an approach.
Job 4 has the largest weight and we should try to give it the largest actual processing time, i.e., $p^*(4) = 6$. This means that job J_4 is processed on one of the restricted machines, e.g., on M_3 in the interval $[0, 4]$, and on the unrestricted machine M_5 for 2 time units within the interval $[4, 7]$, e.g., in the interval $[4, 6]$. To process jobs J_5 and J_6 we have 9 time units: the interval $[0, 4]$ on each machine M_4 and M_5 plus one time unit on M_5 after time 4. Thus, job J_5 can be processed for at most 5 units, e.g., in $[0, 4]$ on machine M_4 and in $[6, 7]$ on machine M_5, while job J_6 is processed in $[0, 4]$ on M_5. See Fig. 1 for the corresponding schedule. We see that $W_1(\lambda) = 5\times6+2\times5+1\times4 = 44$. The total weighted work for this example $W(\lambda) = W_\Delta(\lambda) + W_1(\lambda) = 10 + 44$.

In Example 2, we have just solved the problem of a small size using a common sense reasoning. However, it should be clear that we need a solid mathematical technique to solve the problem in the general case.

We will employ the technique similar to that used in Shioura et al. (2013), based on the ideas of submodular optimization. The problem of maximizing the total weighted work for the conventional jobs can be formulated in terms of

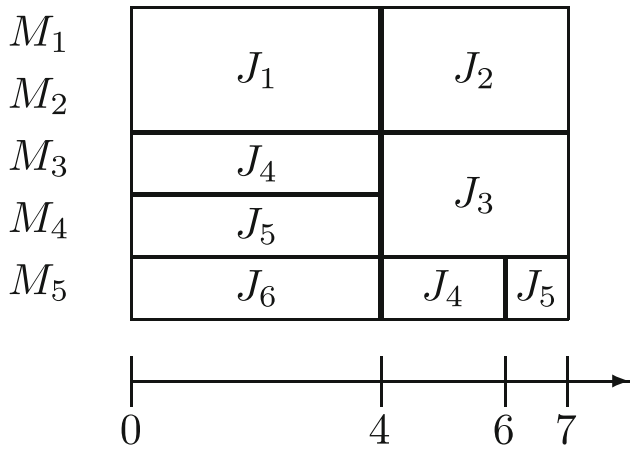

Fig. 1 Schedule for example 2

the linear programming (LP) problem of type (2.3) as

$$\begin{aligned} \text{(1-Job): maximize } & \sum_{j\in N_1} w(j)p(j) && (5.1)\\ \text{subject to } & p(X) \le \varphi(X), && X \subseteq N_1,\\ & l(j) \le p(j) \le u(j), && j \in N_1, \end{aligned}$$

where φ is an appropriate processing capacity function. For a given λ, the optimal value of this LP problem is equal to $W_1(\lambda)$.

To derive the function φ, observe that the required processing capacity depends on the cardinality of a chosen set $X \subseteq N_1$ of jobs. If $0 \le |X| \le m'$, then all these jobs can be processed on the unrestricted machines, at most one for each job. If $m' < |X| < m' + \Delta$, then we can fully use the available capacity on the m' unrestricted machines and additionally a required number of the restricted machines. If $m' + \Delta \le |X| \le n_1$, then we can use all machines with available capacity. Thus, the capacity function can be written as

$$\varphi(X, d) = \begin{cases} D|X|, & \text{if } |X| \le m',\\ Dm' + d(|X| - m'), & \text{if } m' < |X| < m' + \Delta,\\ Dm' + d\Delta, & \text{if } m' + \Delta \le |X|. \end{cases} \quad (5.2)$$

Under the conditions of Example 2, for $|X| = 1$ we have the unrestricted machine to process a single job in the interval $[0, D] = [0, 7]$. On the other hand, for $|X| = 2$ two jobs can be processed on the unrestricted machine in $[0, 7]$ and on one of the restricted machines in $[0, d] = [0, 3]$.

Notice that in (5.2) we stress that the capacity function depends on both, the chosen set X and the small deadline d. Although d is fixed in problem (5.1), for our further purposes it is useful to emphasis the dependence on d.

It is not difficult to prove that φ is a submodular function for each fixed d; see (Shakhlevich & Strusevich, 2008; Shioura et al., 2018) for details of the proof techniques that

can be used for this purpose. Thus, the feasible region for problem (5.1) is a submodular polyhedron intersected with a box.

In our previous work, on multiple occasions we have applied the result proved in Shakhlevich et al. (2009) that an LP problem of the form (5.1) reduces to maximizing the same objective function over a base polyhedron with the rank function defined by

$$\varphi_l^u(X,d) = \min_{Y\subseteq N_1}\{\varphi(Y,d)+u(X\setminus Y) - l(Y\setminus X)\},\ X\subseteq N_1. \tag{5.3}$$

Such a reduction allows us to apply the most well-known result of submodular optimization which says that this problem can be solved by a greedy algorithm, that delivers components $p^*(j)$, $j\in N_1$, of an optimal solution in a closed form.

For the sequence $\sigma = (\sigma(1),\sigma(2),\ldots,\sigma(n_1))$ defined by (3.2), introduce the sets

$$N_t(\sigma) = \{\sigma(1),\ldots,\sigma(t)\},\ 1\le t\le n_1; \tag{5.4}$$

and for completeness, define $N_0(\sigma)=\emptyset$. Then, the components of an optimal solution are given by

$$p^*(\sigma(t)) = \varphi_l^u(N_t(\sigma),d) - \varphi_l^u(N_{t-1}(\sigma),d),\quad t=1,2,\ldots,n_1. \tag{5.5}$$

The overall optimal value of the function is

$$\begin{aligned} W_1(\lambda) &= \sum_{t=1}^{n_1} w(\sigma(t))(\varphi_l^u(N_t(\sigma),d) - \varphi_l^u(N_{t-1}(\sigma),d)) \\ &= \sum_{t=1}^{n_1}(w(\sigma(t)) - w(\sigma(t+1)))\varphi_l^u(N_t(\sigma),d), \end{aligned} \tag{5.6}$$

where $w(\sigma(n_1+1))=0$.

5.2 Concavity of $W_1(\lambda)$

Using formula (5.6), we prove that function $W_1(\lambda)$ is a piecewise-linear concave function. For this purpose, we rewrite functions $\varphi(X,d)$ and $\varphi_l^u(X,d)$, which are functions of the parameter d, as functions of the parameter λ.

We substitute $m' = m-(r+1)\Delta$ into the formula (5.2) and obtain

$$\varphi(X,d) = \begin{cases} D|X|, & \text{if } |X|+(r+1)\Delta\le m,\\ D(m-(r+1)\Delta) & \\ \quad +d(|X|-m+(r+1)\Delta), & \text{if } m<|X|+(r+1)\Delta<m+\Delta,\\ D(m-(r+1)\Delta)+d\Delta, & \text{if } m+\Delta\le|X|+(r+1)\Delta. \end{cases} \tag{5.7}$$

Since this formula is dependent on the parameter r, we write here $\varphi(X,d)=\varphi_r(X,d)$, and define the function $\widetilde{\varphi}(X,\lambda)$ by

$$\widetilde{\varphi}(X,\lambda) = \begin{cases} \varphi_0(X,D-\lambda), & \text{if } 0<\lambda\le D,\\ \varphi_1(X,2D-\lambda), & \text{if } D<\lambda\le 2D,\\ \vdots & \vdots\\ \varphi_{\lfloor m/\Delta\rfloor-1}\ (X,(\lfloor m/\Delta\rfloor)D-\lambda), & \\ \text{if } (\lfloor m/\Delta\rfloor-1)D<\lambda\le\lfloor m/\Delta\rfloor D. & \end{cases}$$

Lemma 4 *For each fixed $X\subseteq N_1$, function $\widetilde{\varphi}(X,\lambda)$ is a piecewise-linear concave function in λ.*

Proof Since

$$\widetilde{\varphi}(X,\lambda) = \varphi_r(X,(r+1)D-\lambda)\quad \text{for } \lambda\in(rD,(r+1)D],$$

it follows from (5.7) that function $\widetilde{\varphi}(X,\lambda)$ is linear in the interval $(rD,(r+1)D]$. In the rest of this proof, we show that for $r=0,1,2,\ldots,\lfloor m/\Delta\rfloor-2$, $\widetilde{\varphi}(X,\lambda)$ is continuous and concave at the point $\lambda=(r+1)D$. For this purpose, we fix an integer r, $r\in\{0,1,2,\ldots,\lfloor m/\Delta\rfloor-2\}$, and show that the following two properties hold:

Property A:

$$\lim_{\lambda\nearrow(r+1)D}\widetilde{\varphi}(X,\lambda) = \lim_{\lambda\searrow(r+1)D}\widetilde{\varphi}(X,\lambda); \tag{5.8}$$

Property B: the slope of $\widetilde{\varphi}(X,\lambda)$ in the interval $\lambda\in(rD,(r+1)D]$ is no smaller than the slope of $\widetilde{\varphi}(X,\lambda)$ in the interval $\lambda\in((r+1)D,(r+2)D]$.

Depending on the cardinality of set X, we have five cases to consider. The cases arise when we assume different positions of $|X|+(r+1)\Delta$ and $|X|+(r+2)\Delta$ regarding m and $m+\Delta$.

Case 1: $|X|+(r+2)\Delta\le m$.

In this case, we have

$$\widetilde{\varphi}(X,\lambda) = D|X|\quad \text{for } \lambda\in(rD,(r+2)D],$$

so that Properties A and B obviously hold.

Case 2: $|X|+(r+1)\Delta<m<|X|+(r+2)\Delta<m+\Delta$.

In this case, we have

$$\begin{aligned} &\widetilde{\varphi}(X,\lambda) = D|X|\quad \text{for } \lambda\in(rD,(r+1)D],\\ &\widetilde{\varphi}(X,\lambda) = D(m-(r+2)\Delta)+((r+2)D-\lambda)\times\\ &\qquad(|X|-m+(r+2)\Delta)\quad \text{for } \lambda\in((r+1)D,(r+2)D]. \end{aligned}$$

Notice that

$$\lim_{\lambda\searrow(r+1)D}\widetilde{\varphi}(X,\lambda) = D|X|,$$

so that Property A holds. Besides, the slope of $\widetilde{\varphi}(X,\lambda)$ is zero for $\lambda \in (rD,(r+1)D]$ and is equal to $-(|X|-m+(r+2)\Delta) < 0$ for $\lambda \in ((r+1)D,(r+2)D]$, so that Property B also holds.

Case 3: $|X|+(r+1)\Delta \le m,\ m+\Delta \le |X|+(r+2)\Delta$.
In this case, $|X|+(r+1)\Delta = m$ and

$$\widetilde{\varphi}(X,\lambda) = D|X| \quad \text{for } \lambda \in (rD,(r+1)D],$$
$$\widetilde{\varphi}(X,\lambda) = D(m-(r+2)\Delta)+((r+2)D-\lambda)\Delta = Dm-\lambda\Delta \quad \text{for } \lambda \in ((r+1)D,(r+2)D].$$

Notice that

$$\lim_{\lambda \searrow (r+1)D} \widetilde{\varphi}(X,\lambda) = Dm-(r+1)\,D\Delta = D|X|,$$

so that Property A holds. Besides, the slope of $\widetilde{\varphi}(X,\lambda)$ is zero for $\lambda \in (rD,(r+1)D]$ and is equal to $-\Delta < 0$ for $\lambda \in ((r+1)D,(r+2)D]$, so that Property B also holds.

Case 4: $m < |X|+(r+1)\Delta < m+\Delta < |X|+(r+2)\Delta$.
In this case, we have

$$\widetilde{\varphi}(X,\lambda) = D(m-(r+1)\Delta)+((r+1)D-\lambda)(|X|-m+(r+1)\Delta) \text{ for } \lambda \in (rD,(r+1)D],$$
$$\widetilde{\varphi}(X,\lambda) = D(m-(r+2)\Delta)+((r+2)D-\lambda)\Delta \text{ for } \lambda \in ((r+1)D,(r+2)D].$$

We have that

$$\lim_{\lambda \nearrow (r+1)D} \widetilde{\varphi}(X,\lambda) = D(m-(r+1)\Delta);$$
$$\lim_{\lambda \searrow (r+1)D} \widetilde{\varphi}(X,\lambda) = D(m-(r+2)\Delta)+D\Delta,$$

so that Property A holds. Besides, the slope of $\widetilde{\varphi}(X,\lambda)$ is $-(|X|-m+(r+1)\Delta)$ for $\lambda \in (rD,(r+1)D]$ and is equal to $-\Delta$ for $\lambda \in ((r+1)D,(r+2)D]$. Since $-(|X|-m+(r+1)\Delta) > -\Delta$ by the assumption that defines this case, we conclude that Property B also holds.

Case 5: $m+\Delta \le |X|+(r+1)\Delta$.
In this case, we have

$$\widetilde{\varphi}(X,\lambda) = D(m-(r+1)\Delta)+((r+1)D-\lambda)\Delta \quad \text{for } \lambda \in (rD,(r+2)D],$$

so that Properties A and B obviously hold.

This concludes the proof of the concavity of function $\widetilde{\varphi}(X,\lambda)$. □

In accordance with (5.3), define

$$\widetilde{\varphi}_l^u(X,\lambda) = \min_{Y \subseteq N_1} \{\widetilde{\varphi}(Y,\lambda)+u(X\setminus Y)-l(Y\setminus X)\}, \quad X \subseteq N_1.$$

Since $\widetilde{\varphi}(Y,\lambda)$ is piecewise-linear concave in λ, for each set $Y \subseteq N_1$ the function $\widetilde{\varphi}(Y,\lambda)+u(X\setminus Y)-l(Y\setminus X)$ is a piecewise-linear concave function in λ, which implies that the function $\widetilde{\varphi}_l^u(X,\lambda)$ is also a piecewise-linear concave function in λ, since the minimum of several piecewise-linear concave functions is also a piecewise-linear concave function. It follows from (5.6) that

$$W_1(\lambda) = \sum_{t=1}^{n_1} (w(\sigma(t))-w(\sigma(t+1)))\widetilde{\varphi}_l^u(N_t(\sigma),\lambda).$$

This shows that $W_1(\lambda)$ is a weighted sum of piecewise-linear concave functions, which implies the following statement.

Lemma 5 *Function $W_1(\lambda)$ is a piecewise-linear concave function.*

6 Computation of $W_1(\lambda)$

In this section, we present an algorithm that for a given λ computes the value of $W_1(\lambda)$ in $O(n_1)$ time under the assumptions made at the beginning of Sect. 3.

Let r be the integer with $\lambda \in (rD,(r+1)D]$. It follows from (5.6) that in order to compute $W_1(\lambda)$ it suffices to compute the function values $\varphi_l^u(N_t(\sigma),d)$, where $d = (r+1)D-\lambda$. We introduce the sets

$$\mathcal{Y}_v = \{Y \subseteq N_1 \mid |Y| = v\},\ 1 \le v \le n_1, \tag{6.1}$$

that contain all subsets of the ground set N_1 with exactly v elements; for completeness we define $\mathcal{Y}_0 = \{\emptyset\}$. Combining (5.2) and (5.3) and using the identity $u(X\setminus Y) = u(X)-u(X\cap Y)$, define

$$\psi'(X,d) = u(X)+\min_{0\le v\le m'} \left\{ Dv-\max_{Y\in\mathcal{Y}_v}\{u(X\cap Y)+l(Y\setminus X)\}\right\}; \tag{6.2}$$
$$\psi''(X,d) = u(X)+Dm'-dm'+\min_{m'<v<m'+\Delta} \left\{ dv-\max_{Y\in\mathcal{Y}_v}\{u(X\cap Y)+l(Y\setminus X)\}\right\}; \tag{6.3}$$
$$\psi'''(X,d) = u(X)+Dm'+d\Delta-\max_{m'+\Delta\le v\le n_1} \left\{ \max_{Y\in\mathcal{Y}_v}\{u(X\cap Y)+l(Y\setminus X)\}\right\}, \tag{6.4}$$

which imply that

$$\varphi_l^u(X,d) = \min\left\{\psi'(X,d),\psi''(X,d),\psi'''(X,d)\right\}. \tag{6.5}$$

By (5.6), to compute the function value $W_1(\lambda)$, we need to compute the smallest of the values $\psi'(X,d)$, $\psi''(X,d)$

and $\psi'''(X,d)$ for $X = N_t(\sigma)$, where $0 \le t \le n_1$. Algorithm ComputeW1 that performs this computation is presented below.

To compute $\psi'(X,d)$ for a given set $X \subseteq N_1$, observe that this function does not depend on d and the minimum in the right-hand side is sought for among the differences between v multiples of D and the sum of v numbers, each being either a u-value or an l -value. It follows from $D \ge u(j) \ge l(j)$, $j \in N_1$, that the minimum in the right-hand side is achieved by $v = 0$, i.e.,

$$\psi'(X,d) = u(X). \tag{6.6}$$

Thus, all required values $\psi'(N_t(\sigma),d), t = 0,1,\ldots,n_1$, can be easily computed. See Step 1 of Algorithm ComputeW1 below.

For $\psi'''(X,d)$, the maximum in the right-hand side is attained by $v = n_1$, so that

$$\begin{aligned}\psi'''(X,d) &= u(X) + Dm' + d\Delta - (u(X) + l(N_1 \setminus X)) \\ &= Dm' + d\Delta - l(N_1 \setminus X),\end{aligned} \tag{6.7}$$

and all required values $\psi'''(N_t(\sigma),d), t = 0,1,\ldots,n_1$, can be easily computed. See Step 2 of Algorithm ComputeW1 below.

We now consider function $\psi''(X,d)$. In the right-hand side of (6.3) the minimum is sought for among the differences of v multiples of d and the sum of v numbers, each equal to some u-value or some l-value, and the minimum is attained if these u- and l-values are greater than d. Formally, for a given set X, define

$$\overline{Y}(X,d) = \{j \in X \mid u(j) > d\} \cup \{j \in N_1 \setminus X \mid l(j) > d\}.$$

Then, we have

$$\begin{aligned}\min_{0 \le v \le n_1} \left\{ dv - \max_{Y \in \mathcal{Y}_v} \{u(X \cap Y) + l(Y \setminus X)\} \right\} &= d \cdot \left|\overline{Y}(X,d)\right| \\ -(u(X \cap \overline{Y}(X,d)) + l(\overline{Y}(X,d) \setminus X)),\end{aligned} \tag{6.8}$$

i.e., for each fixed d, the set $\overline{Y}(X,d)$ is the set-minimizer that is smallest with respect to inclusion. Hence, if $m' < |\overline{Y}(X,d)| < m' + \Delta$, then (6.3) implies that

$$\begin{aligned}\psi''(X,d) = u(X) + Dm' - dm' + d \cdot |\overline{Y}(X,d)| \\ -u(X \cap \overline{Y}(X,d)) - l(\overline{Y}(X,d) \setminus X).\end{aligned}$$

The lemma below shows that if the set $\overline{Y}(X,d)$ does not satisfy the condition $m' < |\overline{Y}(X,d)| < m' + \Delta$, then the function $\psi''(X,d)$ can be ignored in computing $\varphi_l^u(X,d)$.

Lemma 6 *Suppose that either* $\left|\overline{Y}(X,d)\right| \ge m' + \Delta$ *or* $\left|\overline{Y}(X,d)\right| \le m'$ *holds. Then,*

$$\varphi_l^u(X,d) = \min\{\psi'(X,d), \psi'''(X,d)\} \le \psi''(X,d).$$

Proof We show that $\psi''(X,d) \ge \psi'(X,d)$ or $\psi''(X,d) \ge \psi'''(X,d)$ holds. Since $\overline{Y}(X,d)$ satisfies (6.8), it follows that

$$\begin{aligned}&\min_{m' < v < m' + \Delta} \left\{ dv - \max_{Y \in \mathcal{Y}_v} \{u(X \cap Y) + l(Y \setminus X)\} \right\} \\ &\ge d \cdot |\overline{Y}(X,d)| - u(X \cap \overline{Y}(X,d)) - l(\overline{Y}(X,d) \setminus X).\end{aligned} \tag{6.9}$$

Suppose first that $|\overline{Y}(X,d)| \ge m' + \Delta$. By (6.3), (6.7), and (6.9), we have

$$\begin{aligned}&\psi''(X,d) - \psi'''(X,d) \\ &= u(X) - d(m' + \Delta) + l(N_1 \setminus X) \\ &\quad + \min_{m' < v < m' + \Delta} \left\{ dv - \max_{Y \in \mathcal{Y}_v} \{u(X \cap Y) + l(Y \setminus X)\} \right\} \\ &\ge u(X) - d(m' + \Delta) + l(N_1 \setminus X) + d \cdot |\overline{Y}(X,d)| \\ &\quad - u(X \cap \overline{Y}(X,d)) - l(\overline{Y}(X,d) \setminus X) \\ &= d(|\overline{Y}(X,d)| - m' - \Delta) + u(X \setminus \overline{Y}(X,d)) \\ &\quad + l(N_1 \setminus (X \cup \overline{Y}(X,d))) \ge 0,\end{aligned}$$

where the last inequality follows from the assumption $|\overline{Y}(X,d)| \ge m' + \Delta$ and from the non-negativity of $u(j)$ and $l(j)$. Hence, we have $\psi''(X,d) \ge \psi'''(X,d)$.

Suppose now that $|\overline{Y}(X,d)| \le m'$. By (6.3), (6.6), and (6.9), we have

$$\begin{aligned}&\psi''(X,d) - \psi'(X,d) \\ &= Dm' - dm' \\ &\quad + \min_{m' < v < m' + \Delta} \left\{ dv - \max_{Y \in \mathcal{Y}_v} \{u(X \cap Y) + l(Y \setminus X)\} \right\} \\ &\ge Dm' - dm' + d \cdot |\overline{Y}(X,d)| \\ &\quad - u(X \cap \overline{Y}(X,d)) - l(\overline{Y}(X,d) \setminus X) \\ &\ge Dm' - dm' + d \cdot |\overline{Y}(X,d)| - D \cdot |\overline{Y}(X,d)| \\ &= (D - d)(m' - |\overline{Y}(X,d)|) \ge 0,\end{aligned}$$

where the second inequality follows from $l(j) \le u(j) \le D$ for $j \in N$, while the last inequality follows from $d \le D$ and from the assumption $|\overline{Y}(X,d)| \le m'$. Hence, we have $\psi''(X,d) \ge \psi'(X,d)$. □

The lemma above implies that for each $t = 0,1,\ldots,n_1$, in order to determine $\varphi_l^u(N_t(\sigma),d)$, the value of $\psi''(N_t(\sigma),d)$ must be compared to the smaller of $\psi'(N_t(\sigma),d)$ and $\psi'''(N_t(\sigma),d)$ only if $m' < \overline{Y}(N_t(\sigma),d) < m' + \Delta$.

The algorithm for computing the value $W_1(\lambda)$ is presented below.

Algorithm ComputeW1

Step 0. Let r be the integer such that $rD < \lambda \leq (r+1)D$. Set $d := (r+1)D - \lambda$, $m' := m - (r+1)\Delta$.

Step 1. For $t = 0, 1, \ldots, n_1$, set $\psi'_t := U_t$, where U_t is given by (3.3).

Step 2. For $t = 0, 1, \ldots, n_1$, set $\psi'''_t := Dm' + d\Delta - (L_{n_1} - L_t)$, where L_t is given by (3.3).

Step 3. Define

$$\overline{Y_0} := \{j \in N_1 \mid l(j) > d\},$$
$$v_0 := |\overline{Y_0}|,\ Z_0 := dv_0 - l(\overline{Y_0}).$$

For $t = 0, 1, \ldots, n_1 - 1$ do

(a) If $\sigma(t+1) \in \overline{Y_t}$, then set

$$\overline{Y}_{t+1} := \overline{Y}_t;\ v_{t+1} := v_t;\ Z_{t+1} := Z_t + l(\sigma(t+1)) - u(\sigma(t+1)).$$

(b) If $\sigma(t+1) \notin \overline{Y}_t$ and $u(\sigma(t+1)) > d$, then set

$$\overline{Y}_{t+1} := \overline{Y}_t \cup \{\sigma(t+1)\};\ v_{t+1} := v_t + 1;$$
$$Z_{t+1} := Z_t + d - u(\sigma(t+1)).$$

(c) If $\sigma(t+1) \notin \overline{Y}_t$ and $u(\sigma(t+1)) \leq d$, then set

$$\overline{Y}_{t+1} := \overline{Y}_t;\ v_{t+1} := v_t;\ Z_{t+1} := Z_t.$$

Step 4. For $t = 0, 1, \ldots, n_1$, set $\psi''_t := U_t + Dm' - dm' + Z_t$.

Step 5. For $t = 0, 1, \ldots, n_1$, set

$$\varphi_l^u(N_t(\sigma), d) := \begin{cases} \min\{\psi'_t, \psi''_t, \psi'''_t\}, & \text{if } m'+1 \leq v_t \leq m'+\Delta-1, \\ \min\{\psi'_t, \psi'''_t\}, & \text{otherwise.} \end{cases}$$

Step 6. Output the value $\sum_{t=1}^{n_1}(w(\sigma(t)) - w(\sigma(t+1)))\varphi_l^u(N_t(\sigma), d)$.

It is easy to see that the running time of Algorithm ComputeW1 is $O(n_1)$.

Lemma 7 *For any $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, the function value $W_1(\lambda)$ can be computed in $O(n_1)$ time.*

7 Search by major breakpoints

In this section, we describe how to find up to two consecutive major intervals of λ, which is total work of the Δ-jobs, that contain an optimal value of λ^* that maximizes function $W(\lambda)$. The running time of the presented search algorithm is $O(n \log n_\Delta)$.

We know from Sect. 4 that function $W_\Delta(\lambda)$ is concave, and from Sect. 5 that function $W_1(\lambda)$ is concave. Thus, the overall function $W(\lambda)$ is concave, as it is the sum of these two functions. Hence, we have the following properties of an optimal value $\lambda = \lambda^*$ that maximizes the function value $W(\lambda)$:

For any two real numbers λ' and λ'' such that $\lambda' < \lambda''$,

- if $W(\lambda') = W(\lambda'')$, then there exists an optimal λ^* such that $\lambda' \leq \lambda^* \leq \lambda''$, since $W(\lambda) \leq W(\lambda')$ for $\lambda \leq \lambda'$ and $W(\lambda) \leq W(\lambda'')$ for $\lambda \geq \lambda''$;
- if $W(\lambda') > W(\lambda'')$, then there exists an optimal λ^* such that $\lambda^* \leq \lambda''$, since $W(\lambda) \leq W(\lambda'')$ for $\lambda \geq \lambda''$;
- if $W(\lambda') < W(\lambda'')$, then there exists an optimal λ^* such that $\lambda^* \geq \lambda'$, since $W(\lambda) \leq W(\lambda')$ for $\lambda \leq \lambda'$.

This justifies the application of binary search for determining a rough location of the global optimum of function $W(\lambda)$. Such an algorithm is outlined below. The algorithm calls Procedure ComputeMajor described in Sect. 4, which outputs a sorted list of the major breakpoints.

For a given interval $I = [\lambda', \lambda'']$ of λ-values, the *increment* $\delta(I)$ of function $W(\lambda)$ is defined as the difference $W(\lambda'') - W(\lambda')$ of the function values computed at the endpoints of the interval.

Algorithm SearchMajor

Step 1. Call Procedure ComputeMajor to compute the sorted list of the major breakpoints. Suppose that the total number of the found major breakpoints is $g+1$. Denote the corresponding major intervals by $I_1, I_2, \ldots, I_g$.

Step 2. Compute the increments $\delta(I_1)$ and $\delta(I_g)$ of function $W(\lambda)$. If $\delta(I_1) \leq 0$, then output the interval I_1 and stop. If $\delta(I_g) \geq 0$, then output the interval I_g and stop. Otherwise, go to Step 3.

Step 3. Perform binary search among the major intervals with a purpose of finding two consecutive intervals I' and I'', one with a positive increment $\delta(I')$ and the other with a negative increment $\delta(I'')$. Set $k' := 1$ and $k'' := g$.

(a) If $k' + 1 = k''$, then output intervals $I_{k'}$ and $I_{k''}$ and stop. Otherwise, go to Step 3(b).

(b) Set $k := \lfloor \frac{1}{2}(k' + k'') \rfloor$ and compute the increment $\delta(I_k)$ of function $W(\lambda)$.

(c) If $\delta(I_k) = 0$, then output the interval I_k and stop.

If $\delta(I_k) > 0$, then update $k' := k$ and return to Step 3(a).
Otherwise (i.e., $\delta(I_k) < 0$), update $k'' := k$ and return to Step 3(a).

In Algorithm SearchMajor, Steps 1 and 2 can be done in $O(n)$ time by Lemmas 1, 3, and 7. Since the number of major breakpoints is $O(n_\Delta)$ by Lemma 3, the number of iterations of binary search in Step 3 is $O(\log n_\Delta)$. Each iteration in Step 3 can be implemented in $O(n)$ time by Lemmas 1 and 7. Thus, the overall time complexity of Algorithm SearchMajor is $O(n \log n_\Delta)$.

Algorithm SearchMajor outputs either a single major interval or two consecutive major intervals. The endpoints of each of these intervals are major breakpoints of function $W(\lambda)$, and the global maximum of function $W(\lambda)$ is achieved within these intervals. Notice that within each of major intervals the function $W_\Delta(\lambda)$ is linear. In order to find the global maximum of $W(\lambda)$, we need to closely inspect the behavior of the function $W_1(\lambda)$ over each of the found major intervals. As established in Sect. 5, the function $W_1(\lambda)$ is piecewise-linear over each of these intervals, and we need to determine its breakpoints, which we call *minor* breakpoints. The next sections present details for finding and handling minor breakpoints.

8 Minor breakpoints

Consider I, one of the major intervals found by running Algorithm SearchMajor. In what follows we explain how to determine the minor breakpoints of function $W(\lambda)$ within interval I, i.e., the breakpoints of function $W_1(\lambda)$ within I.

Remark The algorithm presented in this section does not find *exactly* the set of all minor breakpoints in I, but rather finds its superset, i.e., the set obtained by the algorithm contains all minor breakpoints and may contain some points that are not minor breakpoints. This is sufficient for our purpose of finding an optimal λ^* that maximizes the function $W(\lambda)$ since λ^* is contained in the set of minor breakpoints and therefore in the set computed by the algorithm.

8.1 Classification of breakpoints

We can represent function $W_1(\lambda)$ by using function $\varphi_l^u(N_t(\sigma), d)$ as follows. The chosen major interval I is contained in the interval $[rD, (r+1)D]$ for some integer r (see (4.4)). Then, we have $d = (r+1)D - \lambda$, and (5.6) implies that

$$W_1(\lambda) = \sum_{t=1}^{n_1} (w(\sigma(t)) - w(\sigma(t+1))) \times \varphi_l^u(N_t(\sigma), (r+1)D - \lambda), \lambda \in I.$$

This shows that all breakpoints of $W_1(\lambda)$ in the major interval I can be obtained by finding all breakpoints of functions $\varphi_l^u(N_t(\sigma), d)$ for $t = 1, 2, \ldots, n_1$ within the interval $[0, D]$. In the remainder of this section, we focus on the latter problem.

Recall that $\varphi_l^u(N_t(\sigma), d)$ can be represented as

$$\varphi_l^u(N_t(\sigma), d) = \min\left\{\psi'(N_t(\sigma), d), \psi''(N_t(\sigma), d), \psi'''(N_t(\sigma), d)\right\}$$

with the functions $\psi'(N_t(\sigma), d)$, $\psi''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ given by (6.6), (6.3), and (6.7), respectively, with $X = N_t(\sigma)$. That is,

$$\psi'(N_t(\sigma), d) = u(N_t(\sigma)), \tag{8.1}$$

$$\psi''(N_t(\sigma), d) = u(N_t(\sigma)) + Dm' - dm' + \min_{m' < v < m' + \Delta} \left\{ dv - \max_{Y \in \mathcal{Y}_v} \{u(N_t(\sigma) \cap Y) + l(Y \setminus N_t(\sigma))\} \right\}, \tag{8.2}$$

$$\psi'''(N_t(\sigma), d) = Dm' + d\Delta - l(N_1 \setminus N_t(\sigma)). \tag{8.3}$$

While the functions $\varphi_l^u(N_t(\sigma), d)$, $\psi'(N_t(\sigma), d)$, $\psi''(N_t(\sigma), d)$, $\psi'''(N_t(\sigma), d)$ are defined on the finite interval $[0, D]$, in the following we extend the domain of the functions to the infinite interval $(-\infty, +\infty)$ to simplify the discussion. For our purpose, we just compute all breakpoints of the function $\varphi_l^u(N_t(\sigma), d)$ and then take those contained in the interval $[0, D]$.

Since the functions $\psi'(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ are linear, the breakpoints of function $\varphi_l^u(N_t(\sigma), d)$ can be classified into four types:

Type 1: breakpoints of $\psi''(N_t(\sigma), d)$;
Type 2: intersection points of $\psi'(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$,
Type 3: intersection points of $\psi'(N_t(\sigma), d)$ and $\psi''(N_t(\sigma), d)$;
Type 4: intersection points of $\psi''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$.

Here, *intersection points* of two functions of the argument d are defined as the values of d for which both functions are equal. It is easy to see that for each integer t, a Type 2 breakpoint is uniquely determined since both $\psi'(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ are linear functions. We will show that for each integer t, Type 3 and Type 4 breakpoints are also uniquely determined.

In our analysis of the minor breakpoints, let $\mathcal{B} = (\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_{2n_1})$ be a non-increasing sequence of the values $l(j)$ and $u(j)$, $j \in N_1$. Such a sequence, which we call

the $\mathcal{B}$-*sequence* can be obtained in $O(n_1 \log n_1)$ time. We assume that the $\mathcal{B}$-sequence is available before starting the computation of minor breakpoints.

For each t, $0 \le t \le n_1$, let $\mathcal{B}_t$ be a sequence that contains the values of $u(j)$ for $j \in N_t(\sigma)$ and the values of $l(j)$ for $j \in N_1 \backslash N_t(\sigma)$. For $z = 1, 2, \ldots, n_1$, let $\beta_z^{(t)}$ denote the z-th largest number in sequence $\mathcal{B}_t$. Notice that the sequence $\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{n_1}^{(t)}$ is a sub-sequence of the $\mathcal{B}$-sequence.

8.2 Type 1 and Type 2 breakpoints

We first consider the Type 1 breakpoints and show that those breakpoints are defined by either u-values or l-values.

Using the sequence $\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{n_1}^{(t)}$, function $\psi''(N_t(\sigma), d)$ of the form (8.2) can be rewritten as

$$\psi''(N_t(\sigma), d) = u(N_t(\sigma)) + Dm' - dm' + \min_{m' < v < m' + \Delta} \left\{ dv - \sum_{z=1}^{v} \beta_z^{(t)} \right\}. \tag{8.4}$$

For each integer v with $m' < v < m' + \Delta$, we define a linear function $\psi_v''(N_t(\sigma), d)$ by

$$\psi_v''(N_t(\sigma), d) = u(N_t(\sigma)) + Dm' - dm' + dv - \sum_{z=1}^{v} \beta_z^{(t)}.$$

Then, (8.4) implies that

$$\psi''(N_t(\sigma), d) = \min_{m' < v < m' + \Delta} \left\{ \psi_v''(N_t(\sigma), d) \right\}.$$

To obtain a more explicit formula for $\psi''(N_t(\sigma), d)$, introduce the intervals

$$I_{m'+\Delta-1}^{(t)} = \left(-\infty, \beta_{m'+\Delta-1}^{(t)} \right],$$
$$I_v^{(t)} = \left[\beta_{v+1}^{(t)}, \beta_v^{(t)} \right), \quad v = m' + \Delta - 2, m' + \Delta - 3, \ldots, m' + 2,$$
$$I_{m'+1}^{(t)} = \left[\beta_{m'+2}^{(t)}, +\infty \right).$$

Lemma 8 *For $t = 0, 1, \ldots, n_1$ and $v = m' + 1, m' + 2, \ldots, m' + \Delta - 1$, if $d \in I_v^{(t)}$ then $\psi''(N_t(\sigma), d) = \psi_v''(N_t(\sigma), d)$. Moreover, the breakpoints of $\psi''(N_t(\sigma), d)$ are given by $\beta_z^{(t)}$, $z = m' + 2, m' + 3, \ldots, m' + \Delta - 1$.*

Note that $\beta_{m'+1}^{(t)}$ is not a breakpoint of $\psi''(N_t(\sigma), d)$ within the interval $I_{m'+1}^{(t)}$, as parameter v in (8.4) takes the minimum value $m' + 1$ for any $d > \beta_{m'+2}^{(t)}$.

Proof Since $\psi''(N_t(\sigma), d)$ is a piecewise-linear concave function, it suffices to show that for each integer $\hat{v} = m' + 1, m' + 2, \ldots, m' + \Delta - 2$, if $d = \beta_{\hat{v}+1}^{(t)}$ then we have

$$\psi''(N_t(\sigma), d) = \psi_{\hat{v}}''(N_t(\sigma), d) = \psi_{\hat{v}+1}''(N_t(\sigma), d). \tag{8.5}$$

To prove the first equality in (8.5), we show that at $d = \beta_{\hat{v}+1}^{(t)}$, for any v, $m' < v < m' + \Delta$, the linear function $\psi_v''(N_t(\sigma), d)$ has the same or larger value than $\psi_{\hat{v}}''(N_t(\sigma), d)$. For $v \in \{m' + 1, m' + 2, \ldots, m' + \Delta - 2\}$, it holds that

$$\psi_v''(N_t(\sigma), d) - \psi_{\hat{v}}''(N_t(\sigma), d) = \left(\beta_{\hat{v}+1}^{(t)} v - \sum_{z=1}^{v} \beta_z^{(t)} \right) - \left(\beta_{\hat{v}+1}^{(t)} \hat{v} - \sum_{z=1}^{\hat{v}} \beta_z^{(t)} \right) = \begin{cases} \sum_{z=\hat{v}+1}^{v} (\beta_{\hat{v}+1}^{(t)} - \beta_z^{(t)}) & \text{if } v \ge \hat{v}, \\ \sum_{z=v+1}^{\hat{v}} (\beta_z^{(t)} - \beta_{\hat{v}+1}^{(t)}) & \text{if } v < \hat{v}. \end{cases} \tag{8.6}$$

Since the values $\beta_z^{(t)}$ are non-increasing with respect to z, we have $\sum_{z=\hat{v}+1}^{v} (\beta_{\hat{v}+1}^{(t)} - \beta_z^{(t)}) \ge 0$ if $v \ge \hat{v}$ and $\sum_{z=v+1}^{\hat{v}} (\beta_z^{(t)} - \beta_{\hat{v}+1}^{(t)}) \ge 0$ if $v < \hat{v}$. Hence, Eq. (8.6) implies that $\psi_v''(N_t(\sigma), d) \ge \psi_{\hat{v}}''(N_t(\sigma), d)$.

The second equality in (8.5) follows immediately from (8.6) in the case $v = \hat{v} + 1$ since

$$\psi_{\hat{v}+1}''(N_t(\sigma), d) - \psi_{\hat{v}}''(N_t(\sigma), d) = \beta_{\hat{v}+1}^{(t)} - \beta_{\hat{v}+1}^{(t)} = 0.$$

□

By the lemma above, all Type 1 breakpoints of $\varphi_l^u(N_t(\sigma), d)$ for $t = 0, 1, 2, \ldots, n_1$ belong to the sequence $\mathcal{B}$ and they can be computed in $O(n_1)$ time.

For $t = 0, 1, \ldots, n_1$, a Type 2 breakpoint of $\varphi_l^u(N_t(\sigma), d)$ is the intersection of two linear functions $\varphi'(N_t(\sigma), d)$ and $\varphi'''(N_t(\sigma), d)$, i.e., the point

$$d = \frac{u(N_t(\sigma)) + l(N_1 \setminus N_t(\sigma)) - Dm'}{\Delta} = \frac{U_t + L_{n_1} - L_t - Dm'}{\Delta};$$

recall the definitions of U_t and L_t in (3.3). Therefore, the Type 2 breakpoints for all t can be also obtained in $O(n_1)$ time.

8.3 Type 3 and Type 4 breakpoints

We first show that for each t, Type 3 and Type 4 breakpoints are uniquely determined.

Lemma 9 *For each $t = 0, 1, \ldots, n_1$, the intersection points of $\psi'(N_t(\sigma), d)$ and $\psi''(N_t(\sigma), d)$, as well as of $\psi''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ are uniquely determined.*

Proof To prove the claim, we use the following fundamental property of a continuous function:

> for a continuous and strictly increasing function $f : \mathbb{R} \to \mathbb{R}$ such that $\lim_{d\to+\infty} f(d) = +\infty$ and $\lim_{d\to-\infty} f(d) = -\infty$, a solution of the equation $f(d) = 0$ is uniquely determined.

We show that the continuous functions $f(d) = \psi''(N_t(\sigma), d) - \psi'(N_t(\sigma), d)$ and $\tilde{f}(d) = \psi'''(N_t(\sigma), d) - \psi''(N_t(\sigma), d)$ satisfy the conditions above.

By (8.1), (8.2), and (8.3), the slope of $\psi'(N_t(\sigma), d)$ is zero, the slope of $\psi'''(N_t(\sigma), d)$ is Δ, and the slope of $\psi''(N_t(\sigma), d)$ is between 1 and $\Delta - 1$. Hence, the functions $f(d) = \psi''(N_t(\sigma), d) - \psi'(N_t(\sigma), d)$ and $\tilde{f}(d) = \psi'''(N_t(\sigma), d) - \psi''(N_t(\sigma), d)$ are strictly increasing and satisfy

$$\lim_{d\to+\infty} f(d) = +\infty, \quad \lim_{d\to-\infty} f(d) = -\infty,$$
$$\lim_{d\to+\infty} \tilde{f}(d) = +\infty, \quad \lim_{d\to-\infty} \tilde{f}(d) = -\infty.$$

□

For $t = 0, 1, \ldots, n_1$, denote by $\gamma^{(t)}$ the (unique) intersection point of $\psi'(N_t(\sigma), d)$ and $\psi''(N_t(\sigma), d)$, i.e., $\gamma^{(t)}$ is a Type 3 breakpoint. We now present an algorithm for finding the Type 3 breakpoints.

The idea of our algorithm is simple. Recall the representation of function $\psi''(N_t(\sigma), d)$ and the definition of the intervals $I_v^{(t)}$ used in Lemma 8. To compute $\gamma^{(t)}$, for each integer v from $m' + \Delta - 1$ down to $m' + 1$ we compute the intersection point $\gamma_v^{(t)}$ of two linear functions $\psi_v''(N_t(\sigma), d)$ and $\psi'(N_t(\sigma), d)$. The intersection point $\gamma_v^{(t)}$ is given as

$$\gamma_v^{(t)} = \frac{1}{v - m'}(\Theta_v^{(t)} - Dm'), \tag{8.7}$$

where

$$\Theta_v^{(t)} = \sum_{z=1}^{v} \beta_z^{(t)}, \qquad t = 0, 1, \ldots, n_1, \ v = m' + 1, m' + 2, \ldots, m' + \Delta - 1, \tag{8.8}$$

i.e., $\Theta_v^{(t)}$ is the sum of v largest elements in sequence $\mathcal{B}_t$. We have $\gamma^{(t)} = \gamma_v^{(t)}$ if the intersection point $\gamma_v^{(t)}$ is contained in the interval $I_v^{(t)}$, and such v always exists.

Algorithm ComputeGamma1

Step 0. Compute $\beta_1^{(0)}, \beta_2^{(0)}, \ldots, \beta_{m'+\Delta-1}^{(0)}$, and $\Theta_{m'+\Delta-1}^{(0)}$. Set $t := 0$.
Step 1. Compute the value $\gamma^{(t)}$ as follows:
Step 1-1. Set $v := m' + \Delta - 1$.
Step 1-2. Compute the intersection point $\gamma_v^{(t)}$ of the two linear functions $\psi_v''(N_t(\sigma), d)$ and $\psi'(N_t(\sigma), d)$ by (8.7).
Step 1-3. If $\gamma_v^{(t)} \in I_v^{(t)}$, then set $\gamma^{(t)} := \gamma_v^{(t)}$ and go to Step 2 (Note: if $v = m' + 1$, then the condition $\gamma_v^{(t)} \in I_v^{(t)}$ must hold).
Otherwise, set $\Theta_{v-1}^{(t)} := \Theta_v^{(t)} - \beta_v^{(t)}$, $v := v - 1$, and go to Step 1-2.
Step 2. If $t = n_1$, then stop. Otherwise, go to Step 3.
Step 3. Compute $\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{m'+\Delta-1}^{(t+1)}$, and $\Theta_{m'+\Delta-1}^{(t+1)}$.
Step 4. Set $t := t + 1$ and go to Step 1.

□

We explain implementation details of the algorithm and analyze its running time.

In Steps 0 and 3, we compute $\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{m'+\Delta-1}^{(t)}$, which can be done easily in $O(n_1)$ time by using the $\mathcal{B}$-sequence since $\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{m'+\Delta-1}^{(t)}$ is a sub-sequence of the $\mathcal{B}$-sequence. The value $\Theta_{m'+\Delta-1}^{(0)}$ is initially computed in $O(n_1)$ time, and then updated in Step 3 in constant time by using the following formula:

$$\Theta_{m'+\Delta-1}^{(t+1)} = \begin{cases} \Theta_{m'+\Delta-1}^{(t)} & \text{if } \beta_{m'+\Delta-1}^{(t)} \geq u(\sigma(t+1)), \\ \Theta_{m'+\Delta-1}^{(t)} - \beta_{m'+\Delta-1}^{(t)} + u(\sigma(t+1)) & \text{if } u(\sigma(t+1)) > \beta_{m'+\Delta-1}^{(t)} > l(\sigma(t+1)), \\ \Theta_{m'+\Delta-1}^{(t)} - l(\sigma(t+1)) + u(\sigma(t+1)) & \text{if } l(\sigma(t+1)) \geq \beta_{m'+\Delta-1}^{(t)}. \end{cases}$$

Steps 1-1, 1-2, and 1-3 can be done in constant time. Since the number of iterations in Step 1 is $O(\Delta)$, Step 1 requires $O(\Delta)$ time in total.

Steps 2 and 4 can be done in constant time.

Since Steps 1–4 are repeated $O(n_1)$ times, the running time of Algorithm ComputeGamma1 is $O((n_1)^2)$.

Now, we present an algorithm for finding the Type 4 breakpoints, which is similar to Algorithm ComputeGamma1 for finding the Type 3 breakpoints. For $t = 0, 1, \ldots, n_1$, we denote by $\alpha^{(t)}$ the (unique) intersection point of $\psi''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$, i.e., $\alpha^{(t)}$ is a Type 4 breakpoint.

To compute $\alpha^{(t)}$, for each integer v from $m' + \Delta - 1$ down to $m' + 1$ we compute the intersection point $\alpha_v^{(t)}$ of two linear functions $\psi_v''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$. The intersection

point $\alpha_v^{(t)}$ is given as

$$\alpha_v^{(t)} = \frac{1}{m' + \Delta - v}(\Upsilon_t - \Theta_v^{(t)}), \tag{8.9}$$

where

$$\Upsilon_t = u(N_t(\sigma)) + l(N_1 \setminus N_t(\sigma)), \qquad t = 0, 1, \ldots, n_1. \tag{8.10}$$

We have $\alpha^{(t)} = \alpha_v^{(t)}$ if the intersection point $\alpha_v^{(t)}$ is contained in the interval $I_v^{(t)}$, and such v always exists.

Algorithm ComputeAlpha1

Step 0. Compute $\beta_1^{(0)}, \beta_2^{(0)}, \ldots, \beta_{m'+\Delta-1}^{(0)}$. Also, compute Υ_0 and $\Theta_{m'+\Delta-1}^{(0)}$ by (8.10) and (8.8), accordingly. Set $t := 0$.

Step 1. Compute the value $\alpha^{(t)}$ as follows:

Step 1-1. Set $v := m' + \Delta - 1$.

Step 1-2. Compute the intersection point $\alpha_v^{(t)}$ of the two linear functions $\psi_v''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ by (8.9).

Step 1-3. If $\alpha_v^{(t)} \in I_v^{(t)}$, then set $\alpha^{(t)} = \alpha_v^{(t)}$ and go to Step 2 (Note: if $v = m'+1$, then the condition $\alpha_v^{(t)} \in I_v^{(t)}$ must hold).
Otherwise, set $\Theta_{v-1}^{(t)} := \Theta_v^{(t)} - \beta_v^{(t)}$, $v := v - 1$, and go to Step 1-2.

Step 2. If $t = n_1$, then stop. Otherwise, go to Step 3.

Step 3. Compute $\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{m'+\Delta-1}^{(t+1)}$. Also, compute $\Theta_{m'+\Delta-1}^{(t+1)}$.

Step 4. Set $\Upsilon_{t+1} := \Upsilon_t + u(\sigma(t+1)) - l(\sigma(t+1))$, $t := t + 1$, and go to Step 1.

□

As in the analysis for Algorithm ComputeGamma1, we can show that the running time of Algorithm ComputeAlpha1 is $O((n_1)^2)$. In Appendix A, we will present a faster algorithm for computing the Type 3 and Type 4 breakpoints.

9 Search by minor breakpoints

Let I^* be a major interval, and suppose that we find (a superset of) all minor breakpoints $\mu_1, \mu_2, \ldots, \mu_s$ contained in the major interval I^*; for convenience, we denote by μ_0 and μ_{s+1} the left and right endpoints of the interval I^*. By the result in Sect. 8, we have $s = O(n_1)$. In this section, we show that some $\mu_k \in I^*$ that maximizes the function value $W(\lambda)$ can be found in $O(n \log n_1)$ time.

The approach of the algorithm described below is essentially the same as the one used in Sect. 7. Since the function $W(\lambda)$ is concave, we have the following properties for an optimal value $\lambda = \lambda^*$ that maximizes the function value $W(\lambda)$ within the interval I^*.

For any consecutive minor breakpoints $\mu' < \mu''$ in the interval I^*, one of the following holds:

- if $W(\mu') = W(\mu'')$, then both of μ' and μ'' are optimal since $W(\lambda) \le W(\mu')$ for $\lambda \le \mu'$, $W(\lambda) = W(\mu') = W(\mu'')$ for $\lambda \in [\mu', \mu'']$, and $W(\lambda) \le W(\mu'')$ for $\lambda \ge \mu''$.
- if $W(\mu') > W(\mu'')$, then there exists an optimal λ^* such that $\lambda^* \le \mu'$ since $W(\lambda) \le W(\mu')$ for $\lambda \ge \mu'$;
- if $W(\mu') < W(\mu'')$, then there exists an optimal λ^* such that $\lambda^* \ge \mu''$ since $W(\lambda) \le W(\mu'')$ for $\lambda \le \mu''$.

Notice that for any consecutive minor breakpoints $\mu' < \mu''$, function $W(\lambda)$ is linear in the interval $[\mu', \mu'']$.

This justifies the application of binary search for determining an optimal $\lambda = \lambda^*$ that maximizes the function $W(\lambda)$ in the interval I^*. Recall the definition of an increment $\delta(I)$ of the function $W(\lambda)$ for an interval $I = [\mu', \mu'']$ of λ-values, as defined in Sect. 7.

Algorithm Minor

Step 1. Find an increasing sequence $\mu_1, \mu_2, \ldots, \mu_s$ of real numbers containing all minor breakpoints of $W(\lambda)$ in interval I^*. Also, let μ_0 and μ_{s+1} be the left and right endpoints of the interval I^*, respectively. The values $\mu_0, \mu_1, \ldots, \mu_{s+1}$ split the interval I^* into intervals $I_1, I_2, \ldots, I_{s+1}$, where $I_k = [\mu_{k-1}, \mu_k]$, $1 \le k \le s+1$.

Step 2. Compute the increments $\delta(I_1)$ and $\delta(I_{s+1})$ of function $W(\lambda)$. If $\delta(I_1) \le 0$, then output the value $\lambda^* = \mu_0$ and stop. If $\delta(I_{s+1}) \ge 0$, then output the value $\lambda^* = \mu_{s+1}$ and stop. Otherwise, go to Step 3.

Step 3 Perform binary search among the found intervals with a purpose of finding two consecutive intervals I' and I'', one with a positive increment $\delta(I')$ and the other with a negative increment $\delta(I'')$. Start with $k' = 1$, $k'' = s + 1$.

(a) If $k' + 1 = k''$, then output the value $\mu_{k'}$ and stop. Otherwise, go to Step 3(b).

(b) Compute $k = \lfloor \frac{1}{2}(k'' + k') \rfloor$ and the increment $\delta(I_k)$ of function $W(\lambda)$.

(c) If $\delta(I_k) = 0$, then output the value μ_k (or μ_{k-1}) and stop.
If $\delta(I_k) > 0$, then update $k' = k$ and return to Step 3(a).
Otherwise (i.e., $\delta(I_k) < 0$), update $k'' = k$ and return to Step 3(a).

In Algorithm Minor, Step 1 can be done in $O((n_1)^2)$ time by the results presented in Sect. 8. Step 2 can be done in $O(n)$ time by Lemmas 1 and 7. The number of iterations of binary search in Step 3 is $O(\log n_1)$. Each iteration in Step 3 can be implemented in $O(n)$ time by Lemmas 1 and 7. Thus, the overall time complexity of Algorithm Minor is $O(n \log n_1 + (n_1)^2)$.

The results presented in this section imply that after the major intervals that contain an optimal value of λ are found, that optimal value can be found in $O(n \log n_1 + (n_1)^2)$ time. This concludes the proof of Theorem 1, that states that the scheduling problem $P|pmtn, p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D| \sum_{j \in N} w(j)x(j)$ can be solved in $O(n \log n + (n_1)^2)$ time.

10 Conclusions

The main result of this paper is an $O(n \log n)$-time algorithm for minimizing total compression cost on identical parallel machines, provided that the processing times of the jobs are controllable and each job either requires one machine or $\Delta > 1$ machines for its processing. Although the problem does not admit a direct reformulation in terms of submodular optimization, some techniques developed in our prior research (Shioura et al., 2018) still appear to be useful; see, e.g., Sect. 5. This paper shows that even in a fairly simple settings, with only two sizes of the jobs, considerable technical efforts are required to design a fast algorithm.

The problems that combine the presence of parallel jobs and controllable processing times have been insufficiently studied. We hope that this work will initiate systematic studies in this area, and that will lead to establishing fairly general methodological principles.

Acknowledgements This work is supported by the project EP/T01461X/1 "Algorithmic Support for Massive Scale Distributed Systems" funded by the Engineering and Physical Sciences Research Council (EPSRC). The first author is partially supported by the Grant 18K11177 of the Japanese Society for the Promotion of Science (JSPS) KAKENHI. The first version of this paper was written when the second author was at the University of Greenwich, London, U.K.

Data Availability No data are associated with this article.

Appendix A: Faster algorithm for finding the Type 3 and Type 4 breakpoints

Algorithm Minor from Sect. 9 locates the optimal value λ^* in a major interval I^* in $O(n \log n + (n_1)^2)$ time. The term $(n_1)^2$ is responsible for finding the Type 3 breakpoints $\gamma^{(0)}, \gamma^{(1)}, \ldots, \gamma^{(n_1)}$ and the Type 4 breakpoints $\alpha^{(0)}, \alpha^{(1)}, \ldots, \alpha^{(n_1)}$ according to the algorithms from Sect. 8.3. Here we present improved algorithms for finding the Type 3 and Type 4 breakpoints and show that each of them runs in $O(n_1 + m \log m)$ time. These results prove Theorem 2, which states that the problem $P|pmtn, p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \le D| \sum_{j \in N} w(j)x(j)$ can be solved in $O(n \log n)$ time.

A1 Monotonicity properties of minor breakpoints

Type 3 and Type 4 breakpoints have the following monotonicity, which will be used in developing an efficient algorithm for computing those breakpoints.

Lemma 10 $\alpha^{(0)} \le \alpha^{(1)} \le \cdots \le \alpha^{(n_1)}$.

Proof For an integer t, $0 \le t < n_1$, we show that $\alpha^{(t)} \le \alpha^{(t+1)}$ holds. By the definitions of $\alpha^{(t)}$ and $\alpha^{(t+1)}$, we have

$$\begin{aligned} \psi''(N_t(\sigma), \alpha^{(t)}) = \psi'''(N_t(\sigma), \alpha^{(t)}), \quad \psi''(N_{t+1}(\sigma), \alpha^{(t+1)}) \\ = \psi'''(N_{t+1}(\sigma), \alpha^{(t+1)}). \end{aligned} \tag{A1.1}$$

By (8.3), we have

$$\begin{aligned} &\psi'''(N_t(\sigma), \alpha^{(t)}) - \psi'''(N_{t+1}(\sigma), \alpha^{(t+1)}) \\ &= \left(Dm' + \alpha^{(t)}\Delta - l(N_1 \setminus N_t(\sigma))\right) \\ &\quad - \left(Dm' + \alpha^{(t+1)}\Delta - l(N_1 \setminus N_{t+1}(\sigma))\right) \\ &= (\alpha^{(t)} - \alpha^{(t+1)})\Delta - l(\sigma(t+1)). \end{aligned} \tag{A1.2}$$

Let $v(t+1), m' < v(t+1) < m' + \Delta$, be an integer such that

$$\begin{aligned} \psi''(N_{t+1}(\sigma), \alpha^{(t+1)}) = u(N_{t+1}(\sigma)) + Dm - \alpha^{(t+1)}m' \\ + \alpha^{(t+1)}v(t+1) - \sum_{z=1}^{v(t+1)} \beta_z^{(t+1)}. \end{aligned} \tag{A1.3}$$

By (8.4), we have

$$\begin{aligned} \psi''(N_t(\sigma), \alpha^{(t)}) \le u(N_t(\sigma)) + Dm - \alpha^{(t)}m' \\ + \alpha^{(t)}v(t+1) - \sum_{z=1}^{v(t+1)} \beta_z^{(t)}. \end{aligned} \tag{A1.4}$$

It follows from (A1.3) and (A1.4) that

$$\begin{aligned}
&\psi''(N_t(\sigma), \alpha^{(t)}) - \psi''(N_{t+1}(\sigma), \alpha^{(t+1)}) \\
&\quad \le \Big(u(N_t(\sigma)) + Dm - \alpha^{(t)}m' \\
&\qquad + \alpha^{(t)}v(t+1) - \sum_{z=1}^{v(t+1)} \beta_z^{(t)}\Big) \\
&\qquad - \Big(u(N_{t+1}(\sigma)) + Dm - \alpha^{(t+1)}m' \\
&\qquad + \alpha^{(t+1)}v(t+1) - \sum_{z=1}^{v(t+1)} \beta_z^{(t+1)}\Big) \\
&\quad = -u(\sigma(t+1)) + (\alpha^{(t)} - \alpha^{(t+1)})(v(t+1) - m') \\
&\qquad - \sum_{z=1}^{v(t+1)} \Big(\beta_z^{(t)} - \beta_z^{(t+1)}\Big). \qquad \text{(A1.5)}
\end{aligned}$$

By (A1.1), (A1.2) and (A1.5), we have

$$\begin{aligned}
0 &= \psi''(N_t(\sigma), \alpha^{(t)}) - \psi''(N_{t+1}(\sigma), \alpha^{(t+1)}) \\
&\quad - \psi'''(N_t(\sigma), \alpha^{(t)}) + \psi'''(N_{t+1}(\sigma), \alpha^{(t+1)}) \\
&\le \Big(-u(\sigma(t+1)) + (\alpha^{(t)} - \alpha^{(t+1)})(v(t+1) - m') \\
&\qquad - \sum_{z=1}^{v(t+1)} \Big(\beta_z^{(t)} - \beta_z^{(t+1)}\Big)\Big) \\
&\qquad - \Big((\alpha^{(t)} - \alpha^{(t+1)})\Delta - l(\sigma(t+1))\Big) \\
&= (\alpha^{(t)} - \alpha^{(t+1)})(v(t+1) - m' - \Delta) \\
&\qquad - \sum_{z=1}^{v(t+1)} \Big(\beta_z^{(t)} - \beta_z^{(t+1)}\Big) - (u(\sigma(t+1)) - l(\sigma(t+1)),
\end{aligned}$$

from which the inequality

$$(\alpha^{(t)} - \alpha^{(t+1)})(v(t+1) - m' - \Delta) \ge \xi_t \qquad \text{(A1.6)}$$

follows, where

$$\xi_t = \sum_{z=1}^{v(t+1)} \Big(\beta_z^{(t)} - \beta_z^{(t+1)}\Big) + (u(\sigma(t+1)) - l(\sigma(t+1)).$$

Below we show that $\xi_t \ge 0$. Recall that the value of $\beta_z^{(t)}$ (respectively, $\beta_z^{(t+1)}$) is the z-th largest number in $\mathcal{B}_t$ (respectively, in $\mathcal{B}_{t+1}$), and the sequence $\mathcal{B}_{t+1}$ is obtained from $\mathcal{B}_t$ by the removal of $l(\sigma(t+1))$ and the insertion of $u(\sigma(t+1))$. Hence, we have to examine the following three possibilities:

Case 1: $l(\sigma(t+1))$ is contained in the sequence $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$, and $u(\sigma(t+1))$ is contained in the sequence $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$.

In this case, we have $\xi_t = 0$, since $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$ is obtained from $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$ by the removal of $l(\sigma(t+1))$ and the insertion of $u(\sigma(t+1))$.

Case 2: $l(\sigma(t+1))$ is not contained in the sequence $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$, but $u(\sigma(t+1))$ is contained in the sequence $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$.

In this case, we have $\xi_t \ge 0$ since $l(\sigma(t+1)) \le \beta_{v(t+1)}^{(t)}$ and $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$ is obtained from $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$ by the removal of $\beta_{v(t+1)}^{(t)}$ and the insertion of $u(\sigma(t+1))$.

Case 3: $l(\sigma(t+1))$ is not contained in the sequence $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$, and $u(\sigma(t+1))$ is not contained in the sequence $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$.

In this case, we have $\xi_t = 0$ since the sequences $(\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{v(t+1)}^{(t)})$ and $(\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{v(t+1)}^{(t+1)})$ coincide.

Thus, in any case, $\xi_t \ge 0$ holds. This, together with (A1.6), implies that $\alpha^{(t)} - \alpha^{(t+1)} \le 0$, since $v(t+1) - m' - \Delta < 0$. □

Lemma 11 $\gamma^{(0)} \le \gamma^{(1)} \le \cdots \le \gamma^{(n_1)}$.

Proof For an integer t, $0 \le t < n_1$, we show that $\gamma^{(t)} \le \gamma^{(t+1)}$ holds. By the definitions of $\gamma^{(t)}$ and $\gamma^{(t+1)}$, we have

$$\begin{aligned}
\psi''(N_t(\sigma), \gamma^{(t)}) &= \psi'(N_t(\sigma), \gamma^{(t)}), \quad \psi''(N_{t+1}(\sigma), \gamma^{(t+1)}) \\
&= \psi'(N_{t+1}(\sigma), \gamma^{(t+1)}). \qquad \text{(A1.7)}
\end{aligned}$$

By (8.1), we have

$$\begin{aligned}
&\psi'(N_t(\sigma), \gamma^{(t)}) - \psi'(N_{t+1}(\sigma), \gamma^{(t+1)}) = u(N_t(\sigma)) \\
&\quad - u(N_{t+1}(\sigma)) = -u(\sigma(t+1)). \qquad \text{(A1.8)}
\end{aligned}$$

Let $\tilde{v}(t)$, $m' < \tilde{v}(t) < m' + \Delta$, be an integer such that

$$\begin{aligned}
\psi''(N_t(\sigma), \gamma^{(t)}) &= u(N_t(\sigma)) + Dm' - \gamma^{(t)}m' \\
&\quad + \gamma^{(t)}\tilde{v}(t) - \sum_{z=1}^{\tilde{v}(t)} \beta_z^{(t)}. \qquad \text{(A1.9)}
\end{aligned}$$

By (8.4), we have

$$\begin{aligned}
&\psi''(N_{t+1}(\sigma), \gamma^{(t+1)}) \le u(N_{t+1}(\sigma)) + Dm' - \gamma^{(t+1)}m' \\
&\quad + \gamma^{(t+1)}\tilde{v}(t) - \sum_{z=1}^{\tilde{v}(t)} \beta_z^{(t+1)}. \qquad \text{(A1.10)}
\end{aligned}$$

It follows from (A1.9) and (A1.10) that

$$\begin{aligned}
&\psi''(N_{t+1}(\sigma), \gamma^{(t+1)}) - \psi''(N_t(\sigma), \gamma^{(t)}) \\
&\quad \le \Big(u(N_{t+1}(\sigma)) + Dm' - \gamma^{(t+1)}m' \\
&\qquad + \gamma^{(t+1)}\tilde{v}(t) - \sum_{z=1}^{\tilde{v}(t)} \beta_z^{(t+1)}\Big) \\
&\qquad - \left(u(N_t(\sigma)) + Dm' - \gamma^{(t)}m' + \gamma^{(t)}\tilde{v}(t) - \sum_{z=1}^{\tilde{v}(t)} \beta_z^{(t)}\right) \\
&\quad = u(\sigma(t+1)) + (\gamma^{(t+1)} - \gamma^{(t)})(\tilde{v}(t) - m') \\
&\qquad - \sum_{z=1}^{\tilde{v}(t)} \left(\beta_z^{(t+1)} - \beta_z^{(t)}\right). \qquad \text{(A1.11)}
\end{aligned}$$

By (A1.7), (A1.8), and (A1.11), we have

$$\begin{aligned}
0 &= \psi''(N_{t+1}(\sigma), \gamma^{(t+1)}) - \psi''(N_t(\sigma), \gamma^{(t)}) \\
&\quad - \psi'(N_{t+1}(\sigma), \gamma^{(t+1)}) + \psi'(N_t(\sigma), \gamma^{(t)}) \\
&\le \Big(u(\sigma(t+1)) + (\gamma^{(t+1)} \\
&\qquad - \gamma^{(t)})(\tilde{v}(t) - m') - \sum_{z=1}^{\tilde{v}(t)} \left(\beta_z^{(t+1)} - \beta_z^{(t)}\right)\Big) \\
&\qquad - u(\sigma(t+1)) \\
&= (\gamma^{(t+1)} - \gamma^{(t)})(\tilde{v}(t) - m') - \sum_{z=1}^{\tilde{v}(t)} \left(\beta_z^{(t+1)} - \beta_z^{(t)}\right),
\end{aligned}$$

from which it follows that

$$(\gamma^{(t+1)} - \gamma^{(t)})(\tilde{v}(t) - m') \ge \sum_{z=1}^{\tilde{v}(t)} \left(\beta_z^{(t+1)} - \beta_z^{(t)}\right). \quad \text{(A1.12)}$$

By the definitions of $\beta_z^{(t)}$ and $\beta_z^{(t+1)}$, we have that $\beta_z^{(t)} \le \beta_z^{(t+1)}$ for each z, which implies that $\sum_{z=1}^{\tilde{v}(t)} \left(\beta_z^{(t+1)} - \beta_z^{(t)}\right) \ge 0$. Since $\tilde{v}(t) - m' > 0$ in (A1.12), we have $\gamma^{(t+1)} - \gamma^{(t)} \ge 0$, as required. □

A2 An improved algorithm

We present a faster algorithm for computing the Type 4 minor breakpoints $\alpha^{(t)}$, $t = 0, 1, \ldots, n_1$; the Type 3 minor breakpoints can be found in a similar way and the details of the corresponding algorithm are omitted.

The idea for the faster algorithm is as follows. Recall that to find the value $\alpha^{(t)}$, Algorithm ComputeAlpha1 computes the values $\alpha_z^{(t)}$ for $z = m' + \Delta - 1, m' + \Delta - 2, \ldots, m' + 1$ and checks whether $\alpha_z^{(t)}$ is contained in the interval $I_z^{(t)}$; if $\alpha_z^{(t)} \in I_z^{(t)}$ then $\alpha^{(t)} = \alpha_z^{(t)}$ holds. We, however, know that $\alpha^{(t)} \ge \alpha^{(t-1)}$ by Lemma 10. This implies that it suffices to check whether one of the following conditions holds:

$$\begin{aligned}
&\alpha_v^{(t)} \in I_v^{(t)} \cap \left[\alpha^{(t-1)}, \beta_v^{(t)}\right], \qquad \alpha_z^{(t)} \in I_z^{(t)}, \\
&\quad z = v - 1, v - 2, \ldots, m' + 1,
\end{aligned}$$

where v is an integer such that $\alpha^{(t-1)} \in I_v^{(t)}$. Besides, instead of computing the values $\beta_z^{(t)}$ for all z, $m' < z < m' + \Delta$, in advance at the beginning of each iteration, we compute the values one by one, when necessary.

For $t = 1, 2, \ldots, n_1$, let $v(t-1)$ denote the maximum integer $v \in \{m' + 1, \ldots, m' + \Delta - 1\}$ such that $\alpha^{(t-1)} \in I_v^{(t-1)}$. To compute $v(t-1)$, define the sequence

$$\mathcal{S} = \left(\beta_z^{(t-1)} \mid 1 \le z \le n_1,\ \beta_z^{(t-1)} \ge \alpha^{(t-1)}\right)$$

and its length $|\mathcal{S}|$. By definition,

$$v(t-1) = \max\left\{|\mathcal{S}|,\ m' + 1\right\}.$$

Similarly, for $t = 1, 2, \ldots, n_1$, denote by $v'(t)$ the maximum integer $v \in \{m' + 1, \ldots, m' + \Delta - 1\}$ such that $\alpha^{(t-1)} \in I_v^{(t)}$. To compute $v'(t)$, define the sequence

$$\mathcal{S}' = \left(\beta_z^{(t)} \mid 1 \le z \le n_1,\ \beta_z^{(t)} \ge \alpha^{(t-1)}\right)$$

and its length $|\mathcal{S}'|$. By definition,

$$v'(t) = \max\left\{|\mathcal{S}'|,\ m' + 1\right\}.$$

Lemma 12 *For $t = 1, 2, \ldots, n_1$, we have $v'(t) \in \{v(t-1), v(t-1) + 1\}$.*

Proof Recall that $v(t-1)$ (respectively, $v'(t)$) is defined via the length of the sequence $\mathcal{S}$ (respectively, of $\mathcal{S}'$). By the definition of the values $\beta_z^{(t-1)}$ and $\beta_z^{(t)}$, we have the following three possibilities for the relationship between the sequences $\mathcal{S}$ and $\mathcal{S}'$.

Case 1: $\mathcal{S}' = \mathcal{S}$;
Case 2: $\mathcal{S}'$ is obtained from $\mathcal{S}$ by the insertion of $u(\sigma(t))$;
Case 3: $\mathcal{S}'$ is obtained from $\mathcal{S}$ by the insertion of $u(\sigma(t))$ and the removal of $l(\sigma(t))$.

In Cases 1 and 3, we have $v'(t) = v(t-1)$, while in Case 2, we have $v'(t) = v(t-1) + 1$, if $v(t-1) = |\mathcal{S}| \ge m' + 1$, or $v'(t) = v(t-1) = m' + 1$, if $v(t-1) = m' + 1 > |\mathcal{S}|$. □

We describe below a faster algorithm for computing $\alpha^{(t)}$. While we compute the value $\alpha^{(0)}$ in the same way as in Algorithm ComputeAlpha1, we compute the values $\alpha^{(t)}$ for $t = 1, 2, \ldots, n_1$ using the idea explained above.

Algorithm ComputeAlpha2

Step 0. Compute $\beta_1^{(0)}, \beta_2^{(0)}, \ldots, \beta_{m'+\Delta-1}^{(0)}$. Also, compute Υ_0 and $\Theta_{m'+\Delta-1}^{(0)}$, by (8.10) and (8.8), accordingly. Set $t := 0$.

Step 1. Compute the values $\alpha^{(0)}$ and $v(0)$ as follows:

Step 1-1. Set $v := m' + \Delta - 1$.

Step 1-2. Compute the intersection point $\alpha_v^{(0)}$ of the two linear functions $\psi_v''(N_0(\sigma), d)$ and $\psi'''(N_0(\sigma), d)$ by (8.9).

Step 1-3. If $\alpha_v^{(0)} \in I_v^{(0)}$, then set $\alpha^{(0)} := \alpha_v^{(0)}$, $v(0) := v$, and go to Step 2 (Note: if $v = m' + 1$, then the condition $\alpha_v^{(0)} \in I_v^{(0)}$ must hold).
Otherwise, set $\Theta_{v-1}^{(0)} := \Theta_v^{(0)} - \beta_v^{(0)}$ and $v := v - 1$, and go to Step 1-2.

Step 2. Set $\Upsilon_1 := \Upsilon_0 + u(\sigma(1)) - l(\sigma(1))$ and $t := 1$. Go to Step 3.

Step 3. Compute the value $\alpha^{(t)}$ as follows:

Step 3-1. Compute the values $v'(t), \beta_{v'(t)}^{(t)}$, and $\Theta_{v'(t)}^{(t)}$ (details will be given later).

Step 3-2. Compute the intersection point $\alpha_{v'(t)}^{(t)}$ of the two linear functions $\psi_{v'(t)}''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ by (8.9).

Step 3-3. If $v'(t) = m' + 1$, or $v'(t) \geq m' + 2$ and $\alpha^{(t-1)} \leq \alpha_{v'(t)}^{(t)} \leq \beta_{v'(t)}^{(t)}$, then set $\alpha^{(t)} := \alpha_{v'(t)}^{(t)}$, $v(t) := v'(t)$, and go to Step 4. Otherwise, set $v := v'(t)$ and go to Step 3-4.

Step 3-4. Compute the value $\beta_{v-1}^{(t)}$ (details will be given later). Set $\Theta_{v-1}^{(t)} := \Theta_v^{(t)} - \beta_v^{(t)}$ and $v := v - 1$.

Step 3-5. Compute the intersection point $\alpha_v^{(t)}$ of the two linear functions $\psi_v''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ by (8.9).

Step 3-6. If $v = m' + 1$, or $v \geq m' + 2$ and $\alpha_v^{(t)} \in I_v^{(t)}$, then set $\alpha^{(t)} := \alpha_v^{(t)}$, $v(t) := v$, and go to Step 4.
Otherwise, go to Step 3-4.

Step 4. If $t = n_1$, then stop. Otherwise, set $\Upsilon_{t+1} := \Upsilon_t + u(\sigma(t+1)) - l(\sigma(t+1))$, $t := t + 1$, and go to Step 3.

It is not difficult to see that the algorithm above computes the values $\alpha^{(t)}$ correctly. The most important part in Algorithm ComputeAlpha2 is the computation of values $v'(t)$, $\beta_{v'(t)}^{(t)}$, and $\Theta_{v'(t)}^{(t)}$ in Step 3-1 and $\beta_{v-1}^{(t)}$ in Step 3-4. It is easy to compute these values in $O(n_1)$ time, as in Algorithm ComputeAlpha1, which leads to the same running time as before. To reduce the running time, we use a heap $\mathcal{H}$, which, at the beginning of Step 3, is a multi-set

$$\{\beta_z^{(t)} \mid 1 \leq z \leq n_1,\ \beta_z^{(t)} \geq \alpha^{(t-1)}\}.$$

This implies the following properties:

at the beginning of Step 3,

- $v'(t)$ is equal to $\max\left\{|\mathcal{H}|, m' + 1\right\}$, where $|\mathcal{H}|$ is the size of $\mathcal{H}$,
- $\Theta_{v'(t)}^{(t)}$ is the total sum of the values in $\mathcal{H}$,
- $\beta_{v'(t)}^{(t)}$ is the minimum value in $\mathcal{H}$.

Hence, the values $v'(t)$ and $\Theta_{v'(t)}^{(t)}$ can be obtained in constant time by maintaining the size of $\mathcal{H}$ and the total sum of the values in $\mathcal{H}$. The value $\beta_{v'(t)}^{(t)}$ can be obtained in constant time as well.

To compute the value $\beta_{v-1}^{(t)}$ in Step 3-4, we delete the minimum value from $\mathcal{H}$ so that the new minimum value in $\mathcal{H}$ is equal to $\beta_{v-1}^{(t)}$. This can be done in $O(\log n_1)$ time.

After computing the value $\alpha^{(t)}$ in Step 3, we need to update the heap $\mathcal{H}$, which can be done easily as follows. At this point, $\mathcal{H}$ is represented by the multi-set

$$\{\beta_z^{(t)} \mid 1 \leq z \leq n_1,\ \beta_z^{(t)} \geq \alpha^{(t)}\},$$

which should be updated to

$$\{\beta_z^{(t+1)} \mid 1 \leq z \leq n_1,\ \beta_z^{(t+1)} \geq \alpha^{(t)}\}.$$

Since

$$\{\beta_1^{(t+1)}, \beta_2^{(t+1)}, \ldots, \beta_{n_1}^{(t+1)}\} = \{\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_{n_1}^{(t)}\} \setminus \{l(\sigma(t+1))\} \cup \{u(\sigma(t+1))\},$$

it follows that if $l(\sigma(t+1)) \geq \alpha^{(t)}$ then the value $l(\sigma(t+1))$ is currently in $\mathcal{H}$ and it should be deleted. Additionally, if $u(\sigma(t+1)) \geq \alpha^{(t)}$, the value $u(\sigma(t+1))$ is added to $\mathcal{H}$. In this way, we can update the heap $\mathcal{H}$ in $O(\log n_1)$ time.

We give a detailed implementation of Algorithm ComputeAlpha2, where Steps 0 and 1 are the same as before and omitted.

Step 2. Set $\Upsilon_1 := \Upsilon_0 + u(\sigma(1)) - l(\sigma(1))$. Let $\mathcal{H}$ be the heap which is represented by the multi-set $\{\beta_z^{(0)} \mid 1 \leq z \leq n_1,\ \beta_z^{(0)} \geq \alpha^{(0)}\}$.
If $l(\sigma(1)) \geq \alpha^{(0)}$, then delete the value $l(\sigma(1))$ from $\mathcal{H}$.
If $u(\sigma(1)) \geq \alpha^{(0)}$, then add the value $u(\sigma(1))$ to $\mathcal{H}$.
Set $t := 1$ and compute $v'(1) := \max\left\{|\mathcal{H}|, m' + 1\right\}$, the sum $\Theta_{v'(1)}^{(1)}$ of the elements in $\mathcal{H}$ and its minimum element $\beta_{v'(1)}^{(1)}$. Go to Step 3.

Step 3. Using the current heap $\mathcal{H}$ and the values $v'(t) = \max\left\{|\mathcal{H}|, m' + 1\right\}$, $\Theta_{v'(t)}^{(t)}$ equal to the total sum elements of $\mathcal{H}$ and the minimum element $\beta_{v'(t)}^{(t)}$, compute the value $\alpha^{(t)}$ as follows:

Step 3-1. Compute the intersection point $\alpha_{v'(t)}^{(t)}$ of the two linear functions $\psi_{v'(t)}''(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ by (8.9).

Step 3-2. If $v'(t) = m' + 1$, or $v'(t) \geq m' + 2$ and $\alpha^{(t-1)} \leq \alpha^{(t)}_{v'(t)} \leq \beta^{(t)}_{v'(t)}$, then set $\alpha^{(t)} := \alpha^{(t)}_{v'(t)}$, $v(t) := v'(t)$, and go to Step 4. Otherwise, set $v := v'(t)$ and go to Step 3-3.

Step 3-3. Determine the interval I with the endpoints being the two smallest elements in the heap (to be used in Step 3-5). To do this, store the minimum value in $\mathcal{H}$, which is $\beta^{(t)}_v$, as c'. Delete that minimum value c' from $\mathcal{H}$ and update $\Theta^{(t)}_{v-1} := \Theta^{(t)}_v - c'$. Store the minimum value in the updated heap $\mathcal{H}$, which is $\beta^{(t)}_{v-1}$, as c'' and define $I = [c', c'']$. Set $v := v - 1$.

Step 3-4. Compute the intersection point $\alpha^{(t)}_v$ of the two linear functions $\psi''_v(N_t(\sigma), d)$ and $\psi'''(N_t(\sigma), d)$ by (8.9).

Step 3-5. If $v = m' + 1$, or $v \geq m' + 2$ and $\alpha^{(t)}_v \in I$, then set $\alpha^{(t)} := \alpha^{(t)}_v$, $v(t) := v$, and go to Step 4.
Otherwise, go to Step 3-3.

Step 4. If $t = n_1$, then stop. Otherwise, go to Step 5.

Step 5. Set $\Upsilon_{t+1} := \Upsilon_t + u(\sigma(t+1)) - l(\sigma(t+1))$. If $l(\sigma(t+1)) \geq \alpha^{(t)}$, then delete the value $l(\sigma(t+1))$ from $\mathcal{H}$. If $u(\sigma(t+1)) \geq \alpha^{(t)}$, then add the value $u(\sigma(t+1))$ to $\mathcal{H}$. For the obtained heap, appropriately update the size, the total sum and the minimum element.
Set $t := t + 1$ and go to Step 3.

We analyze the time required to each steps. As in Algorithm ComputeAlpha1, Steps 0 and 1 can be done in $O(n_1)$ time and Step 4 requires $O(1)$ time. Step 2 requires $O(n_1 \log n_1)$ time for initializing the heap $\mathcal{H}$, and Step 5 requires $O(\log n_1)$ time for deletion/insertion in the heap $\mathcal{H}$.

We now analyze the time required in Step 3 for each $t = 1, 2, \ldots, n_1$. It is easy to see that each of Steps 3-1 – 3-5 can be done in $O(\log n_1)$ time. The number of iterations of Steps 3-3, 3-4 and 3-5 is equal to $v'(t) - v(t)$. Hence, Step 3 for each t requires $O((v'(t) - v(t) + 1) \log n_1)$ time. This implies that the total running time of Algorithm ComputeAlpha2 is

$$O(n_1 + \sum_{v=1}^{n_1} (v'(t) - v(t) + 1) \log n_1) = O(n_1 \log n_1 + \sum_{v=1}^{n_1} (v'(t) - v(t)) \log n_1) = O(n_1 \log n_1),$$

where the last equality follows from the lemma below.

Lemma 13 $\sum_{v=1}^{n_1} (v'(t) - v(t)) \leq 2n_1$.

Proof By Lemma 12, we have $v'(t) \leq v(t-1) + 1$ for each $t = 1, 2, \ldots, n_1$. It follows that

$$\sum_{v=1}^{n_1} (v'(t) - v(t)) \leq \sum_{v=1}^{n_1} (v(t-1) + 1 - v(t)) = n_1 + v(0) - v(n_1) \leq 2n_1,$$

where the last inequality is by $v(0) \leq n_1$ and $v(n_1) \geq 0$. □

This concludes the proof of Theorem 2, stating that the problem $P|pmtn, p(j) = u(j) - x(j), size(j) \in \{1, \Delta\}, C(j) \leq D| \sum_{j \in N} w(j)x(j)$ can be solved in $O(n \log n)$ time.

References

Błażewicz, J., Drabowski, M., & Węglarz, J. (1986). Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers, C-35*, 389–393.

Drozdowski, M. (2009). *Scheduling for Parallel Processing*. London: Springer.

Fujishige, S. (2005). *Submodular functions and optimization*, 2nd ed. Annals of Discrete Mathematics, 58.

Horn, W. A. (1974). Some simple scheduling algorithms. *Naval Research Logistics Quarterly, 21*, 177–185.

Jansen, K., & Mastrolilli, M. (2004). Approximation schemes for parallel machine scheduling problems with controllable processing times. *Computers and Operations Research, 31*, 1565–1581.

Jansen, K., & Porkolab, L. (2003). Computing optimal preemptive schedules for parallel tasks: Linear programming approaches. *Mathematical Programming Series A, 95*, 617–630.

Katoh, N., & Ibaraki, T. (1998). Resource allocation problems. In D.-Z. Du & P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization* (Vol. 2, pp. 159–260). Dordrecht: Kluwer.

Kononov, A., & Kovalenko, Y. (2020). Approximation algorithms for energy-efficient scheduling of parallel jobs. *Jouirnal of Scheduling, 23*, 693–709.

Kononov, A., & Kovalenko, Y. (2020). Makespan minimization for parallel jobs with energy constraint. *Lecture Notes in Computer Science, 12095*, 289–300.

Li, K. (1999). Analysis of the list scheduling algorithm for precedence constrained parallel tasks. *Journal of Combinatorial Optimization, 3*, 73–88.

Li, K. (2012). Energy efficient scheduling of parallel tasks on multiprocessor computers. *Journal of Supercomputing, 60*, 223–247.

Lopes, R. V., & Menasce, D. (2016). A taxonomy of job scheduling on distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems, 27*, 3412–3428.

MAGMA. (2022). 2.5.4 Web page (Matrix Algebra for GPU and Multicore Architectures). Retrieved January 19 https://icl.cs.utk.edu/projectsfiles/magma/doxygen/index.html.

McCormick, S. T. (1999). Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Operations Research, 47*, 744–756.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science, 12*, 1–12.

PETSc. (2022). Web page (Portable Extensible Toolkit for Scientific Computation). Retrieved January 19 https://petsc.org/release/

ScaLAPACK. (2022). Web page (Scalable Linear Algebra PACKage). Retrieved January 7 http://www.netlib.org/scalapack/

Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Berlin: Springer.

Sha, L., Abdelzaher, T., Årzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., & Mok, A. K. (2004). Real time scheduling theory: a historical perspective. *Real-Time Systems, 28*, 101–155.

Shabtay, D., & Steiner, G. (2007). A survey of scheduling with controllable processing times. *Discrete Applied Mathematics, 155*, 1643–1666.

Shakhlevich, N. V., & Strusevich, V. A. (2008). Preemptive scheduling on uniform parallel machines with controllable job processing times. *Algorithmica, 51*, 451–473.

Shakhlevich, N. V., Shioura, A., & Strusevich, V. A. (2009). Single machine scheduling with controllable processing times by submodular optimization. *International Journal of Foundations of Computer Science, 20*, 247–269.

Shioura, A., Shakhlevich, N. V., & Strusevich, V. A. (2013). A submodular optimization approach to bicriteria scheduling problems with controllable processing times on parallel machines. *SIAM Journal on Discrete Mathematics, 27*, 186–204.

Shioura, A., Shakhlevich, N. V., & Strusevich, V. A. (2015). Decomposition algorithms for submodular optimization with applications to parallel machine scheduling with controllable processing times. *Mathematical Programming Series A, 153*, 495–534.

Shioura, A., Shakhlevich, N. V., & Strusevich, V. A. (2016). Application of submodular optimization to single machine scheduling with controllable processing times subject to release dates and deadlines. *INFORMS Journal on Computing, 28*, 148–161.

Shioura, A., Shakhlevich, N. V., & Strusevich, V. A. (2017). Machine speed scaling by adapting methods for convex optimization with submodular constraints. *INFORMS Journal on Computing, 29*, 724–736.

Shioura, A., Shakhlevich, N. V., & Strusevich, V. A. (2018). Preemptive models of scheduling with controllable processing times and of scheduling imprecise computation: A review of solution approaches. *European Journal of Operational Research, 266*, 795–818.