Machine Speed Scaling by Adapting Methods for Convex Optimization with Submodular Constraints

Akiyoshi Shioura
Natalia V. Shakhlevich
Vitaly A. Strusevich

Discussion Paper No. 2015-08
June 18, 2015

Department of Social Engineering
Graduate School of Decision Science and Technology
Tokyo Institute of Technology

Machine Speed Scaling by Adapting Methods for Convex Optimization with Submodular Constraints

Akiyoshi Shioura
Department of Social Engineering,
Tokyo Institute of Technology, Tokyo 152-8550, Japan

Natalia V. Shakhlevich
School of Computing, University of Leeds, Leeds LS2 9JT, U.K.

Vitaly A. Strusevich
Department of Mathematical Sciences,
University of Greenwich, Old Royal Naval College,
Park Row, London SE10 9LS, U.K.

Abstract

In this paper, we propose a new methodology for the speed scaling problem based on its link to scheduling with controllable processing times and submodular optimization. It results in faster algorithms for traditional speed scaling models, characterized by a common speed/energy function. In addition, it handles efficiently the most general models with job-dependent speed/energy functions, with a single and multiple machines, which to the best of our knowledge have not been addressed prior to this study. In particular, the general version of the single-machine case is solvable by the new technique in $O(n^2)$ time.

Key words: convex optimization, submodular constraints, single machine scheduling, parallel machine scheduling, energy minimization, controllable processing times

AMS subject classifications. 90C27, 90B35, 90C05

1 Introduction

Scheduling models with variable machine speeds have been studied since the 1980s; see, e.g., [19]. They were re-introduced in the 1990s in the context of energy efficiency of battery operated portable computing devices; see [38]. In those models, processors can work at different voltage/frequency levels, achieving a lower level of energy consumption at the cost of performing computation at a slower rate. The introduction of multi-processor computer systems with processors having changeable speeds has led to further developments in processors' power management. The topic has become particularly important in recent years with increased importance of energy saving demands.

Informally, in speed scaling problems it is required to determine the processing speed of each job either on a single machine or on parallel machines. The speeds are selected in such a way that (i) the cost of speed changing, often understood as energy needed to

maintain a certain speed, is minimized, and (ii) the actual processing time of each job allows its processing within a given time window.

It is widely recognized that the 1995 paper by Yao et al. [38] provides a fundamental algorithmic technique for speed scaling for the most basic model with a single processor. For almost 20 years the $O(n^3)$-time YDS algorithm has remained the main item of reference in the area, with the number of citations exceeding 1200 according to Google Scholar. See Appendix for a description of that algorithm and discussions, including its faster implementations.

The multi-processor version of the problem received attention quite recently, see [4, 5, 6]. The fastest strongly polynomial-time algorithm proposed in [4] solves repeatedly a series of the max-flow problems and requires $O(n^2h(n))$ time, where $h(n)$ is the time complexity for computing the maximum flow in a layered graph with $O(n)$ nodes, which results into an $O\left(n^5\right)$-time algorithm. The link to the max-flow problem is also exploited in [6], however the running time of the resulting algorithm is not strongly polynomial.

In our study, we provide a new insight into the underlying model of the speed scaling problem (SSP) by establishing its link to optimization of a convex function over submodular constraints, which results into a new methodological framework for handling the problem. Applying powerful tools of submodular optimization we achieve faster algorithms for the single-processor and multi-processors cases, with time complexities $O(n^2)$ and $O(n^3)$, respectively.

The proposed methodology makes it possible to address a more general version of the SSP in comparison to those previously studied. While traditionally it is assumed that the energy consumption functions are identical for all jobs, in reality heterogeneous jobs may differ in their energy characteristics (e.g., due to their different read/write characteristics, the sizes of input/output files, the usage of internal and external memory, etc.). We demonstrate that the most general SSP with job-dependent energy consumption can be solved by the submodular optimization techniques in $O(n^2)$ and $O(n^4)$ time for the single-machine and multi-machine cases, respectively. To the best of our knowledge, these are the first results for this general type of the speed scaling model, and the running times compare favorably to those earlier available for solving the SSP with job-independent cost functions.

Notice that the need to consider individual energy models for tasks dependent on their computation intensity or data intensity is widely recognized in the computing community. Practitioners have now started to include in their models more realistic features of the effect of speed scaling. Increasing processor's speed can speed up the computation part of the job, keeping the overheads of read/write operations unchanged. Therefore the energy usage function depends on job characteristics related to the job splitting into a computation part and an input/output part, see, e.g., the survey [37] for an overview of memory bounded codes and [7, 8] for an example of a job-dependent energy function. Despite their importance in applications, the enhanced models of this type have not been addressed from the mathematical point of view.

Formally, in the SSP, the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed either on a single machine M_1 or on parallel machines $M_1, M_2, \ldots, M_m$, where $m \geq 2$. Each job $j \in N$ is given a *release date* $r(j)$, before which it is not available, and a *deadline* $d(j)$, by which its processing must be completed, and its processing volume or size $w(j)$. The value of $w\,(j)$ can be understood as the actual processing time of job j, provided that the speed $s(j)$ of its processing is set equal to 1. In the processing of any job, *preemption* is allowed, so that the processing can be interrupted on any machine at any time and resumed later, possibly on another machine (in the case of parallel machines). It is not allowed to process a job on

more than one machine at a time, and a machine processes at most one job at a time.

The actual processing time $p(j)$ of a job $j \in N$ depends on the speed of the processor which may change over time. In the SSP literature, the power consumption of a machine operating at speed s is proportional to s^3, or in general is described by a convex non-decreasing function $f(s)$. Given a schedule with a specified allocation of jobs to machines and fixed time intervals for processing jobs or their parts, the energy is calculated as power integrated over time. Due to the convexity of f, the power is minimized if each job j is processed with a fixed speed $s(j) \geq 1$, which does not change during the whole processing of a job; see, e.g., [4]. This property also holds if energy consumption functions are different for different jobs. Thus, the actual processing time of job j is equal to $p(j) = w(j)/s(j)$ and the total cost of processing job j is equal to $(w(j)/s(j))f_j(s(j))$, where $f_j(s(j))$ is the cost of keeping the processing speed of job j to be equal to $s(j)$ for one time unit; each function is convex non-decreasing.

In the SSP, the goal is to find an assignment of speeds to jobs such that

(i) the energy consumption is minimized, and

(ii) a feasible schedule (with no job j processed outside the time interval $[r(j), d(j)]$) exists.

The corresponding cost function is defined as

$$F = \sum_{j=1}^{n} \frac{w(j)}{s(j)} f_j(s(j)). \tag{1}$$

Notice that the prior research on the SSP focuses on minimizing a simpler function

$$\Phi = \sum_{j=1}^{n} \frac{w(j)}{s(j)} f(s(j)), \tag{2}$$

in which the speed cost function f is a convex function, common to all jobs.

In a broad sense, the SSP belongs to the area of scheduling models in which a decision-maker is able to control processing parameters. One type of such models, known as *scheduling models with controllable processing times* appears to be especially relevant to the SSP. Scheduling problems of the latter type have been actively studied since the 1980s; see surveys [28, 30]. To demonstrate the link between the problems with controllable processing times and the SSP, below we give a description of the former model for a machine environment similar to that of SSP.

In the model with controllable processing times (CPT), the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed with preemption either on a single machine M_1 or on parallel machines $M_1, M_2, \ldots, M_m$, where $m \geq 2$. Each job j has a release date $r(j)$ and a deadline $d(j)$. A decision needs be made about the actual duration $p(j)$ of a job: it should belong to a given interval $[l(j), w(j)]$. Such a decision results in *compression* of the longest processing time $w(j)$ down to $p(j)$, and the value $z(j) = w(j) - p(j)$ is called the *compression amount* of job j. Compression may decrease the completion time of each job j but incurs additional cost. The purpose is to find the actual processing times such that a feasible schedule exists and the total compression cost $\sum_{j \in N} \alpha(j) z(j)$ is minimized, where $\alpha(j)$ is the cost of compressing job j by one time unit.

The SSP and scheduling problems with CPT are similar; however, they are based on principally different types of control of the actual processing times, and involve different

objective functions. Still, there are several aspects that make the formulated problems with CPT relevant to the SSP. As we demonstrate in this paper, efficient CPT algorithms can be used as subroutines for solving more complex SSP problems (see Section 5 which makes use of an algorithm from [17] for solving a single machine problem with controllable processing times to minimize the total compression time $\sum_{j\in N} z(j)$). Most importantly, unlike the previous purpose-built techniques with a schedule-based reasoning, in our study we consider both types of models, SSP and CPT, as optimization problems with submodular constraints. This 'step change' research allows us to develop a common toolkit for solving scheduling problems of a similar nature. The success of this new methodology for the CPT models has been demonstrated in a series of papers written by our team [31, 32, 33, 34, 35]. As a result, powerful methods of submodular optimization have been used to develop and justify the fastest available algorithms for both single criterion and bicriteria problems with CPT. What we see as a methodological contribution of this paper is the development of a general framework for handling the SSP. We establish links between the SSP on one hand, and the flow problems and submodular optimization problems with non-linear objective functions. This allows us to come up with the faster available methods not by designing purpose-built algorithms, but rather by adapting the existing flow and submodular optimization techniques.

In this paper, we reformulate the SSP as the problem of minimizing function F of the form (1) on parallel machines as a min-cost max-flow problem with a non-linear convex separable objective function; see Section 2. The latter problem is then linked to a non-linear convex minimization problem under submodular constraints, which can be solved by adapting a decomposition algorithm of [11]; see Section 3. In Sections 4 and 5, we show how to implement the decomposition algorithm in such a way that the original SSP is solvable in $O\left(n^4\right)$ time on parallel machines and in $O\left(n^2\right)$ time on a single machine. In the multi-machine case with the objective function Φ we rely on a non-trivial result in [26, 27] to reduce the problem to the minimization problem with a separable quadratic objective, which allows the SSP to be solved in $O\left(n^3\right)$ time.

2 Reduction of Speed Scaling Problems to Minimizing Flow Cost

Given a set $N = \{1, 2, \ldots, n\}$ of jobs to be processed on either a single machine M_1 or on m parallel machines $M_1, M_2, \ldots, M_m$, where $m \geq 2$, consider the speed scaling problem (SSP, for short). For each job $j \in N$, we are given

- $w(j)$, volume of computation of job j, i.e., its processing time at speed equal to 1;
- $r(j)$, the release date;
- $d(j)$, the deadline;
- $f_j(s(j))$, the cost of keeping the processing speed of job j to be equal to $s(j)$ for one time unit.

It is required to minimize a function of the form

$$F = \sum_{j=1}^{n} \frac{w(j)}{s(j)} f_j(s(j)). \tag{3}$$

We can rewrite the problem with the decision variables $p(j) = w(j)/s(j)$, where $p(j)$ is understood as an actual processing time of job $j \in N$. The objective function F becomes

$$\hat{F} = \sum_{j=1}^{n} p(j) f_j \left(\frac{w(j)}{p(j)} \right). \tag{4}$$

This function has to be minimized over all feasible values of $p(j)$. We reformulate the resulting problem as a min-cost max-flow problem in a bipartite network with a non-linear convex objective. For completeness, below we introduce the concepts related to flows in networks, mainly relying on the book [1].

Introduce the following generic bipartite network $G = (V, A)$. Here the node set $V = \{s, t\} \cup N \cup W$ consists of a source s, a sink t and two sets N and W. The set A of arcs is defined as $A = A^s \cup A^0 \cup A^t$, where A_s is the set of arcs that go from the source s to nodes in N, A_t is the set of arcs that go from the nodes in W to the sink t, and A^0 is the set of arcs that go from nodes in N to nodes in W.

The capacity of an arc $(u, v) \in A$ is denoted by $\mu(u, v)$, which can be infinite for some arcs. We say that $x : A \to \mathbb{R}$ is a *feasible s-t flow* in G if it satisfies the flow balance constraint

$$\sum_{u \in V:\,(u,v) \in A} x(u, v) = \sum_{v \in V:\,(u,v) \in A} x(u, v)$$

for all nodes $v \in V \setminus \{s, t\}$ and the capacity constraint $0 \le x(u, v) \le \mu(u, v)$ for all arcs $(u, v) \in A$. The *value* of the flow x is the total flow $\sum_{j \in N} x(s, j)$ on the arcs that leave the source (or, equivalently, the total flow $\sum_{i \in W} x(i, t)$ on the arcs that enter the sink). A *maximum flow* is a feasible s-t flow with the maximum value. In the *max-flow* problem it is required to find a maximum flow.

A partition (S, T) of the node set V is called a *cut*. Given a cut (S, T), introduce the set of arcs

$$A(S, T) = \{(u, v) \in A \mid u \in S,\ v \in T\}$$

and define the *capacity* of the cut as

$$\mu(S, T) = \sum_{(u,v) \in A(S,T)} \mu(u, v),$$

A cut (S, T) is called an *s-t cut* if $s \in S$ and $t \in T$. A *minimum s-t cut* is an s-t cut of the smallest capacity.

Theorem 1 *Let $x : A \to \mathbb{R}$ be a feasible s-t flow and (S, T) be an s-t cut.*
(i) If x is a maximum flow and (S, T) is a minimum s-t cut, then $\sum_{j \in N} x(s, j) = \mu(S, T)$ holds. Moreover, $x(u, v) = \mu(u, v)$ holds for every $(u, v) \in A(S, T)$.
(ii) If $\sum_{j \in N} x(s, j) = \mu(S, T)$, then x is a maximum flow and (S, T) is a minimum s-t cut.

The statement (i) in Theorem 1 is the well-known max-flow min-cut theorem.

In the *min-cost* flow problem, each arc $(u, v) \in A$ is associated with a cost function $c_{(u,v)}(x(u, v))$ for flow $x(u, v)$. It is required to find a feasible s-t flow of a given value that has the smallest cost. In this paper, we mainly will be interested in the *min-cost max-flow* problem, i.e., the problem of finding the maximum flow of the smallest cost.

Given an instance of the SSP, we can associate it with a variant of network G. Divide the interval $[\min_{j \in N} r(j), \max_{j \in N} d(j)]$ into subintervals by using the release dates $r(j)$ and

the deadlines $d(j)$ for $j \in N$ as break-points. Let $\tau_0, \tau_1, \ldots, \tau_\gamma$, where $1 \leq \gamma \leq 2n-1$, be the increasing sequence of distinct numbers in the list $(r(j), d(j) \mid j \in N)$. Introduce the intervals $I_h = [\tau_{h-1}, \tau_h]$, $1 \leq h \leq \gamma$, and define the set of all intervals $W = \{I_h | 1 \leq h \leq \gamma\}$. Denote the length of interval I_h by $\Delta_h = \tau_h - \tau_{h-1}$. Interval I_h is *available* for processing job j if $r(j) \leq \tau_h$ and $d(j) \geq \tau_{h+1}$. For a job j, denote the set of the available intervals by $\Gamma(j)$, where

$$\Gamma(j) = \{I_h \in W \mid I_h \subseteq [r(j), d(j)]\}. \tag{5}$$

For $X \subseteq N$, define the set of all intervals available for precessing the jobs of set X as

$$\Gamma(X) = \bigcup_{j \in X} \Gamma(j). \tag{6}$$

For $X \subseteq N$ and $h = 1, 2, \ldots, \gamma$, we denote by $\eta(X, h)$ the number of jobs in X that can be processed in the interval I_h, i.e.,

$$\eta(X, h) = |\{j \in X \mid I_h \in \Gamma(j)\}|. \tag{7}$$

Notice that for $X, Y \subseteq N$ such that $X \cap Y = \emptyset$, we have

$$\eta(X \cup Y, h) = \eta(X, h) + \eta(Y, h). \tag{8}$$

Introduce the following variant of the generic bipartite network $G = (V, A)$, which we denote by G_∞. The node set is given by $V = \{s, t\} \cup N \cup W$, where N is the set of job nodes and W is the set of interval nodes, i.e., $W = \{I_1, I_2, \ldots, I_\gamma\}$. The arc set A is given as $A = A^s \cup A^0 \cup A^t$, where

$$\begin{aligned} A^s &= \{(s, j) \mid j \in N\}, \\ A^0 &= \{(j, I_h) \mid j \in N,\ I_h \in \Gamma(j)\}, \\ A^t &= \{(I_h, t) \mid h = 1, 2, \ldots, \gamma\}, \end{aligned}$$

so that the source is connected to each job node, each interval node is connected to the sink, and each job node is connected to the nodes associated with the available intervals. We define the arc capacities as follows:

$$\begin{aligned} &\mu(s, j) = +\infty, && (s, j) \in A^s, \\ &\mu(j, I_h) = \Delta_h, && (j, I_h) \in A^0, \\ &\mu(I_h, t) = m\Delta_h, && (I_h, t) \in A^t. \end{aligned}$$

See Figure 1 for an illustration.

As independently shown in [14] and [18], the problem of verifying whether there exists a feasible schedule with fixed processing times $p(j)$, $j \in N$, can be translated in terms of the network flow problem.

Lemma 1 (cf. [14, 18]) *Let $\mathbf{p} = (p(1), \ldots, p(n))$ be an n-dimensional vector with positive components. A feasible schedule for processing the jobs of set N on m identical parallel machines (or on a single machine if $m = 1$) such that job $j \in N$ has the actual processing time of $p(j)$ exists if and only if there exists a feasible s-t flow $x : A \to \mathbb{R}$ in network G_∞ such that $x(s, j) = p(j)$ for all $j \in N$.*

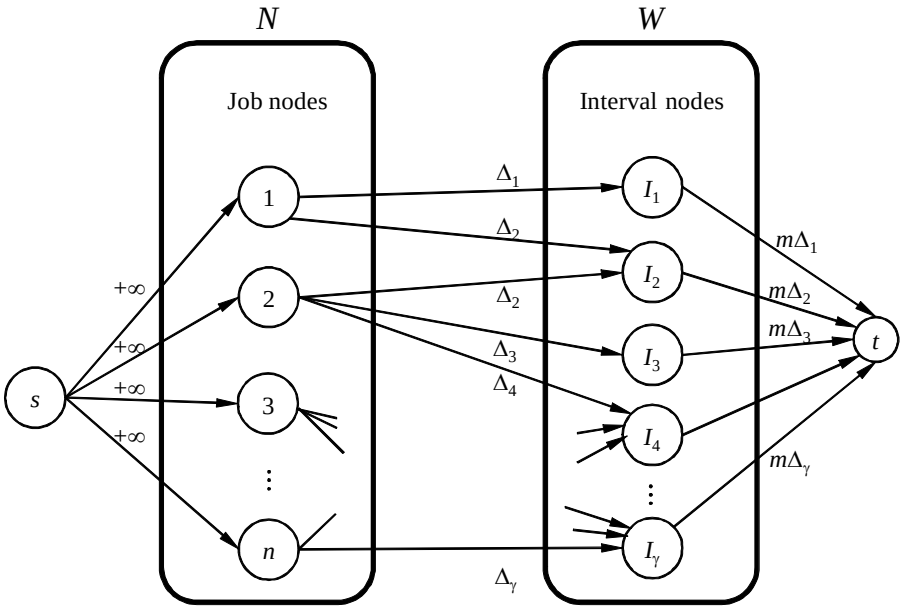

Figure 1: Network $G_\infty = (V, A)$

For a network with a set of nodes V, an algorithm developed by Karzanov [20] finds a maximum flow in $O(|V|^3)$ time. Since $|N| = n$ and $|W| \leq 2n$, Karzanov's algorithm checks the existence of a feasible schedule on m parallel machines in $O(n^3)$ time.

A feasible flow $x(j, I_h)$ on arc (j, I_h) defines for how long job j is processed in the time interval I_h. On a single machine, a feasible flow easily translates into a feasible schedule and vice versa, since there is a one-to-one correspondence between the flow incoming into an interval node I_h and durations of jobs processed within the corresponding time intervals by a single machine. In the case of m identical parallel machines, the link between a feasible flow and a feasible schedule is less evident. To know the flow values $x(j, I_h)$ is insufficient to define a schedule. We need a linear time algorithm by McNaughton [24] to find a feasible preemptive schedule for each interval I_h, and then the overall schedule can be found as a concatenation of these schedules.

In the SSP to minimize function $\hat{F}$ of the form (4), the actual processing times $p(j)$ are not given but are in fact decision variables. Let $x(s, j)$ denote the amount of flow on an arc (s, j), $j \in N$. Define the associated cost function $c_{(s,j)}$ of that flow as

$$c_{(s,j)}(x(s,j)) = x(s,j) f_j \left(\frac{w(j)}{x(s,j)} \right),$$

which is a non-decreasing function with respect to x. The cost of flow on each remaining arcs is set to zero. Then the SSP reduces to finding a maximum flow x^* in network G_∞ that minimizes the total cost $\sum_{j \in N} c_{(s,j)}(x^*(s,j))$. Given a minimum-cost maximum-flow x^*, the optimal processing times of the SSP are given by $p(j) = x^*(s,j)$, $j \in N$, and optimal speeds are $s(j) = w(j)/p(j)$. Note that the proposed model implies that $s\,(j)$ can take any values, so that for $s(j) > 1$ processing is sped up, while for $s(j) < 1$ it is slowed down in comparison with a standard speed of 1.

The derived min-cost max-flow problem has a separable convex nonlinear objective function. A similar formulation can be found in several papers on the SSP; see, e.g., [9] for the

most recent reference. However, unlike most of prior research, we explore a link between network flow problems and submodular optimization. These issues are discussed in the next section.

3 Links to Submodular Optimization

In order to make the paper self-contained, we briefly describe the necessary concepts related to submodular optimization and establish its links to the network flow problems and scheduling problems of interest. Unless stated otherwise, we follow the comprehensive monographs [12] and [29].

Let $N = \{1, 2, \ldots, n\}$ be a ground set, where n is a positive integer, and 2^N denote the family of all subsets of N. For a subset $X \subseteq N$, let $\mathbb{R}^X$ denote the set of all vectors $\mathbf{p}$ with real components $p(j)$, where $j \in X$. For two vectors $\mathbf{p} = (p(1), p(2), \ldots, p(n)) \in \mathbb{R}^N$ and $\mathbf{q} = (q(1), q(2), \ldots, q(n)) \in \mathbb{R}^N$, we write $\mathbf{p} \leq \mathbf{q}$ if $p(j) \leq q(j)$ for each $j \in N$. Given a set $U \subseteq \mathbb{R}^N$, a vector $\mathbf{p} \in U$ is called *maximal* in U if there exists no vector $\mathbf{q} \in U$ such that $\mathbf{p} \leq \mathbf{q}$ and $p(j) < q(j)$ for some $j \in N$. For a vector $\mathbf{p} \in \mathbb{R}^N$, define $p(Y) = \sum_{j \in Y} p(j)$ for every set $Y \in 2^N$.

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

$$\varphi(X \cup Y) + \varphi(X \cap Y) \leq \varphi(X) + \varphi(Y) \tag{9}$$

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 the *rank function* of that system. For a submodular system $(2^N, \varphi)$, define two polyhedra

$$P(\varphi) = \{\mathbf{p} \in \mathbb{R}^N \mid p(X) \leq \varphi(X), X \in 2^N\}, \tag{10}$$

$$B(\varphi) = \{\mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P(\varphi),\ p(N) = \varphi(N)\}, \tag{11}$$

called a *submodular polyhedron* and a *base polyhedron*, respectively, associated with the submodular system. Notice that $B(\varphi)$ represents the set of all maximal vectors in $P(\varphi)$.

A submodular system $\big(2^N, \varphi\big)$ is called a *polymatroid*, provided that the rank function φ is monotone, i.e., $\varphi(X) \leq \varphi(Y)$ for every $X, Y \in 2^N$ with $X \subseteq Y$. The *polymatroid polyhedron* associated with a polymatroid $(2^N, \varphi)$ is given by

$$P_{(+)}(\varphi) = \{\mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P(\varphi),\ \mathbf{p} \geq 0\};$$

in this case we refer to φ as a *polymatroid rank* function. Note that for a polymatroid rank function φ, all vectors in the base polyhedron $B(\varphi)$ are nonnegative, and the base polyhedron $B(\varphi)$ coincides with the set of all maximal vectors in a polymatroid $P_{(+)}(\varphi)$.

Consider the bipartite network G_∞ described in Section 2. We define a polyhedron

$$P = \{\mathbf{p} \in \mathbb{R}^N_+ \mid \exists \text{ feasible } s\text{-}t \text{ flow } x : A \to \mathbb{R} \text{ in } G_\infty \text{ with } p(j) = x(s, j),\ j \in N\}.$$

It is known (see, e.g., [25, Lemma 4.1], [12, Section 2.2], and [16]) that such a polyhedron is a polymatroid polyhedron. Further, all possible maximum flows can be characterized as a base polyhedron $B(\varphi)$ with a polymatroid rank function $\varphi : 2^N \to \mathbb{R}$ given by

$$\varphi(X) = \max\left\{ \sum_{j \in X} y(s, j) \;\middle|\; y \text{ is a feasible } s\text{-}t \text{ flow in } G_\infty \right\}, \quad X \subseteq N, \tag{12}$$

i.e.,

$$B(\varphi) = \left\{ \mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P,\ p(N) = \varphi(N) \right\}. \tag{13}$$

We see from (12) and from the definition of the network G_∞ that the value $\varphi(X)$ is explicitly given as

$$\varphi(X) = \sum_{h=1}^{\gamma} \min\{m, \eta(X,h)\} \cdot \Delta_h, \quad X \subseteq N, \tag{14}$$

where $\eta(X,h)$ is defined by (7). In particular, for the case of a single machine, i.e., for $m = 1$,

$$\varphi(X) = \sum_{I_h \in \Gamma(X)} \Delta_h, \quad X \subseteq N, \tag{15}$$

where $\Gamma(X)$ is given by (6).

In scheduling terms, the value in the right-hand side of (14) specifies the total duration of all time intervals available for processing the jobs of set X or needed for processing the jobs of set X, whichever is smaller. Note that the polymatroid rank function φ can also be represented as

$$\varphi(X) = \max \left\{ \sum_{j \in N} y(s,j) \;\middle|\; y \text{ is a feasible } s\text{-}t \text{ flow in } G_\infty^X \right\}, \quad X \subseteq N, \tag{16}$$

where G_∞^X is a network which differs from G_∞ only by the capacities of the arcs entering the nodes $j \in N \setminus X$: in order to exclude those nodes from consideration, $\mu(s,j)$ are set to 0 for them.

Recall that in Section 2, the SSP has been reduced to the min-cost max-flow problem in network G_∞. Thus, in terms of submodular optimization, the SSP can be reformulated as

$$\begin{aligned} \text{SSP}: \quad \text{minimize} \quad & \sum_{j=1}^{n} p(j) f_j\left(\frac{w(j)}{p(j)}\right) \\ \text{subject to} \quad & \mathbf{p} \in B(\varphi), \end{aligned} \tag{17}$$

with the polymatroid rank function φ defined by (12) with respect to network G_∞. The problem (17) falls into the category of problems of minimizing convex separable functions with a polymatroid constraint:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} h_j\left(p\left(j\right)\right) \\ \text{subject to} \quad & \mathbf{p} \in B(\varphi), \end{aligned} \tag{18}$$

where $h_j(\cdot)$ is a convex function, and $B(\varphi)$ is a base polyhedron associated with a polymatroid rank function φ. In particular, for $h_j(p(j)) = p(j) f_j(w(j)/p(j))$, problem (18) coincides with problem (17). Recall that every $\mathbf{p} \in B(\varphi)$ is a nonnegative vector since φ is a polymatroid rank function.

To solve the problem (18), we can adapt a decomposition algorithm from [15] (see also [12, Section 8.2]). A description of a variant of that algorithm that suits our purposes is given below.

Algorithm F-Decomp

Step 1. Find an optimal solution $\mathbf{b} \in \mathbb{R}^N$ of the following "relaxed" problem with a single constraint:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{j\in N} h_j(p(j)) \\ \text{subject to} & p(N) = \varphi(N), \\ & p(j) \geq 0, \qquad j \in N. \end{array}$$

Step 2. Find a maximal vector $\mathbf{q} \in \mathbb{R}^N$ satisfying the following condition:

$$q(X) \leq \varphi(X),\ X \in 2^N, \qquad 0 \leq q(j) \leq b(j),\ j \in N.$$

Step 3. Find a nonempty set $Y_* \subseteq N$ such that

$$q(Y_*) = \varphi(Y_*). \tag{19}$$

Step 4. If $Y_* = N$, then output the vector $\mathbf{q}$ and stop. Otherwise, go to Step 5.

Step 5. Find an optimal solution $\mathbf{p_1} \in \mathbb{R}^{Y_*}$ of the following problem:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{j\in Y_*} h_j(p(j)) \\ \text{subject to} & p(X) \leq \varphi(X), \quad X \in 2^{Y_*}, \\ & p(Y_*) = \varphi(Y_*). \end{array}$$

Step 6. Find an optimal solution $\mathbf{p_2} \in \mathbb{R}^{N\setminus Y_*}$ of the following problem:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{j\in N\setminus Y_*} h_j(p(j)) \\ \text{subject to} & p(X) \leq \varphi(X \cup Y_*) - \varphi(Y_*), \quad X \in 2^{N\setminus Y_*}, \\ & p(N \setminus Y_*) = \varphi(N) - \varphi(Y_*). \end{array}$$

Step 7. Output the direct sum $\mathbf{p}^* = \mathbf{p_1} \oplus \mathbf{p_2} \in \mathbb{R}^N$ and stop.

Notice that the subproblems to be solved in Steps 1, 5 and 6 share a common structure and can be written in the following form:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{j\in H} h_j(p(j)) \\ \text{subject to} & p(X) \leq \psi(X), \quad X \in 2^H, \\ & p(H) = \psi(H), \end{array} \tag{20}$$

where $\psi : 2^H \to \mathbb{R}$ is a submodular function with $\psi(\emptyset) = 0$ given by

$$\psi(X) = \varphi(X \cup K) - \varphi(K), \quad X \in 2^H, \tag{21}$$

with $H, K \subseteq N$ such that $H \cap K = \emptyset$. Hence, the original problem can solved recursively.

We now present the explicit representation of the function ψ for problem (20).

Take a set $X \subseteq H$ and a set K such that $H \cap K = \emptyset$. Substituting the definition (14) of φ into (21) we obtain:

$$\psi(X) = \varphi(X \cup K) - \varphi(K) = \sum_{h=1}^{\gamma} \{\min\{m, \eta(X \cup K, h)\} - \min\{m, \eta(K, h)\}\} \Delta_h.$$

First, if $I_h \notin \Gamma(X)$ for some h, $1 \leq h \leq \gamma$, then $\eta(X \cup K, h) = \eta(K, h)$ and the equality

$$\min\{m, \eta(X \cup K, h)\} = \min\{m, \eta(K, h)\} \tag{22}$$

holds. Also, if $I_h \in \Gamma(X)$ but $\eta(K, h) \geq m$, then again (22) holds. In either case, the corresponding term does not contribute to $\psi(X)$.

Thus, we only need to consider intervals of the set

$$W(X, K) = \{I_h \mid I_h \in \Gamma(X),\ \eta(K, h) < m\}, \tag{23}$$

namely those intervals I_h, which are suitable for processing the jobs from X ($I_h \in \Gamma(X)$) and are not fully used by the jobs from K ($\eta(K, h) < m$). For an interval $I_h \in W(X, K)$ we have

$$\begin{aligned} \min\{m, \eta(X \cup K, h)\} - \min\{m, \eta(K, h)\} &= \min\{m, \eta(X \cup K, h)\} - \eta(K, h) \\ &= \min\{m - \eta(K, h), \eta(X, h)\}, \end{aligned}$$

where the last equality is due to (8). Thus,

$$\psi(X) = \sum_{I_h \in W(X,K)} \min\{m - \eta(K, h), \eta(X, h)\} \Delta_h. \tag{24}$$

In particular, if $m = 1$ then $W(X, K) = \{I_h \mid I_h \in \Gamma(X),\ I_h \notin \Gamma(K)\}$ and

$$\psi(X) = \sum_{I_h \in \Gamma(X \cup K) \setminus \Gamma(K)} \Delta_h = \sum_{I_h \in \Gamma(X) \setminus \Gamma(K)} \Delta_h. \tag{25}$$

Remark 1 *A set Y_* that satisfies (19) is called a tight set. In a version of the algorithm presented by Fujishige [12], the author recommends that a set found in Step 3 is a maximal (and, therefore, unique) tight set. In each iteration, such a set is used to decompose the current problem. At the same time, it is mentioned in [12] that the requirement of the maximality of the tight set is not crucial. This is why in our description of Algorithm F-Decomp we only insist that Y_* is tight.*

In fact, in the scheduling applications discussed in Sections 4 and 5, we describe how to find a set Y_* which is not only tight, but satisfies a stronger condition

$$q(Y_*) = \varphi(Y_*);\ q(j) = b(j),\ j \in N \setminus Y_*.$$

Notice that a set that satisfies this condition is used in the decomposition algorithm for maximizing a linear function over a submodular polyhedron intersected with a box; see [35], where such a set is called an instrumental set.

Remark 2 *Step 1 involves minimization of a non-linear function. In order to estimate the running time of Algorithm F-Decomp we need an assumption on a possible implementation of that step. It is easy to verify that vector $\mathbf{b} \in \mathbb{R}^N$ found in Step 1, is such that*

$$\frac{dh_j(b(j))}{dp(j)} = \lambda, \quad j \in N,$$

for some λ. We assume that Step 1 can be implemented in $O(n)$ time. This is, for example, true if the power consumption function for job j is of the form $f_j(s(j)) = a(j)s(j)^c$, where $c > 1$ is a constant and $a(j)$ is a job-dependent coefficient that differentiates computation-intensive and data-intensive jobs. In this case

$$h_j(p(j)) = \frac{a(j)w(j)^c}{p(j)^{c-1}}, \quad j \in N$$

and the solution to Step 1 is given by

$$b(j) = \frac{a(j)^{1/c}w(j)\varphi(N)}{\sum_{j\in N} a(j)^{1/c}w(j)}, \quad j \in N.$$

This generalizes the most common case studied in the speed scaling literature with a job-independent power consumption function of the form $f_j(s(j)) = s(j)^c$, where $c > 1$ is a constant. Notice that the case of $c = 3$ corresponds to the well-known cubic root rule for CMOS devices: the speed is approximately equal to the cubic root of the power, or equivalently $f_j(s(j)) = s(j)^3$.

Algorithm F-Decomp admits the following interpretation in scheduling terms. The value $\varphi(X)$ for $X \subseteq N$ specifies the total duration of all time intervals available for processing the jobs of set X. Thus, for the relaxed problem in Step 1, the found values of $b(j), j \in N$, can be understood as actual processing times of jobs such that their total duration $p(N) = b(N)$ is equal to the total duration $\varphi(N)$ of all available intervals. In the case of job-independent speed cost functions (i.e., the speed cost function becomes Φ of the form (2)), this is achieved by processing the jobs at the common speed defined as the total work requirement of all jobs $w(N)$ divided by the total length $\varphi(N)$ of available intervals, i.e., each job j is processed at the same speed $s(j) = w(N)/\varphi(N)$.

For the original SSP, the values of $b(j)$ are not necessarily feasible durations for all jobs or for some subsets of jobs. The required feasible values $q(j)$, $j \in N$, are found in Step 2. The tight set Y_* found in Step 3 identifies a set of jobs with the total duration equal to the total capacity of all intervals available for processing these jobs. In other words, for each job $j \in Y_*$ its actual duration cannot be further extended due to the insufficient processing capacity.

In the subsequent sections, we explain implementation details of the steps of Algorithm F-Decomp in the case of the speed scaling problems on identical parallel machines and on a single machine. In fact, we only need to focus on Steps 2 and 3 which arise when solving the subproblem (20). Step 2 can be interpreted as finding an optimal solution q_* to the following auxiliary linear programming problem:

$$\begin{array}{lll} \text{(LP)}: & \text{Maximize} & \sum_{j\in H} q(j) \\ & \text{subject to} & q(X) \le \psi(X),\ X \in 2^H, \\ & & 0 \le q(j) \le b(j),\ j \in H, \end{array} \tag{26}$$

while Step 3 requires finding a set Y_* with $q_*(Y_*) = \psi(Y_*)$.

Note that the number of subproblems generated by Algorithm F-Decomp is at most $2n-1$ and therefore Steps 2 and 3 are performed $O(n)$ times. Hence, under the assumption made in Remark 2 regarding the time complexity of Step 1, the overall running time of Algorithm F-Decomp is $O(n \cdot T_{23}(n))$, where $T_{23}(h)$ denotes the time required for Steps 2 and 3 with h decision variables.

4 Solving SSP on Parallel Machines

We start with the case of parallel machines. Consider problem (20), where function $\psi : 2^H \rightarrow \mathbb{R}$ is given by (21) with $H, K \subseteq N$ such that $H \cap K = \emptyset$, and the speed cost functions f_j are job-dependent. We show that for solving (20), Steps 2 and 3 of Algorithm F-Decomp can be implemented in $O(n^3)$ time.

Recall that for a set X of jobs, a meaningful interpretation of $\psi(X)$ is the total length of the time intervals available for processing the jobs of set $X \cup K$ after the intervals for processing the jobs of set K have been completely used up. Hence, the function ψ can be represented by a modified version of the network G_∞ in a way similar to (12) for function φ.

Let $\widetilde{G}_\infty(H, K) = (\widetilde{V}, \widetilde{A})$ be a subgraph of G_∞ induced by the set of nodes $\widetilde{V} = \{s, t\} \cup H \cup \widetilde{W}$, where $\widetilde{W} = W(H, K)$ is defined in accordance with (23). Thus, network $\widetilde{G}_\infty(H, K)$ contains the job nodes associated with the jobs of set H and the interval nodes associated with intervals of set $\widetilde{W} = W(H, K)$, which are suitable for processing the jobs from H and are not fully used by the jobs of set K. The set of arcs of $\widetilde{G}_\infty(H, K)$ is given as

$$\widetilde{A} = \{(s, j) \mid j \in H\} \cup \{(j, I_h) \mid j \in H,\ I_h \in \Gamma(j) \cap \widetilde{W}\} \cup \{(I_h, t) \mid I_h \in \widetilde{W}\},$$

and the arc capacities are as follows

$$\begin{aligned}
&\widetilde{\mu}(s, j) = +\infty, \quad (s, j) \in \widetilde{A},\\
&\widetilde{\mu}(j, I_h) = \Delta_h, \quad (j, I_h) \in \widetilde{A},\\
&\widetilde{\mu}(I_h, t) = \min\{m - \eta(K, h), \eta(H, h)\}\Delta_h, \quad (I_h, t) \in \widetilde{A}.
\end{aligned}$$

Note that $\widetilde{\mu}(I_h, t)$ are defined in accordance with (24).

The lemma below shows that for any $X \subseteq H$ the corresponding value of $\psi(X)$ is defined as a max-flow value in $\widetilde{G}_\infty(H, K)$.

Lemma 2 *Given sets $H, K \subseteq N$ such that $H \cap K = \emptyset$, the equality*

$$\psi(X) = \max\left\{ \sum_{j \in X} y(s, j) \;\middle|\; y \text{ is a feasible } s\text{-}t \text{ flow in } \widetilde{G}_\infty(H, K) \right\} \tag{27}$$

holds for each $X \subseteq H$.

Proof: Introduce network $\widetilde{G}^X_\infty$, which differs from $\widetilde{G}_\infty(H, K)$ only by the zero capacities set on the arcs entering the nodes $j \in H \setminus X$. Formally, the arc capacities of $\widetilde{G}^X_\infty$ are denoted by $\widetilde{\mu}^X(u, v)$ and defined by

$$\widetilde{\mu}^X(u, v) = \begin{cases} 0, & \text{if } (u, v) = (s, j) \text{ with } j \in H \setminus X,\\ \widetilde{\mu}(u, v), & \text{otherwise.} \end{cases}$$

Clearly, the maximum flow value $\max\left\{ \sum_{j \in H} y(s, j) \;\middle|\; y \text{ is a feasible } s\text{-}t \text{ flow in } \widetilde{G}^X_\infty \right\}$ is equal to the right-hand side of (27).

Let ζ be the value of the right-hand side of (24). To prove the lemma it suffices to demonstrate that the maximum flow in $\widetilde{G}^X_\infty$ is equal to ζ. Therefore, in the remainder of this proof, we consider network $\widetilde{G}^X_\infty$ instead of $\widetilde{G}_\infty$.

We first explain how to construct a feasible s-t flow in $\widetilde{G}_{\infty}^{X}$ with a value equal to ζ. Then we show that the capacity of the corresponding s-t cut in $\widetilde{G}_{\infty}^{X}$ is also ζ. Thus, by Theorem 1 (ii) the constructed flow is a maximum flow, and it is of the required value ζ.

Split the interval nodes $\widetilde{W}$ into two subsets, $\widetilde{W}_{\text{big}}$ and $\widetilde{W}_{\text{small}}$. Set $\widetilde{W}_{\text{big}}$ consists of intervals I_h which can accommodate all $\eta(X,h)$ jobs that can be processed in I_h. On other other hand, $\widetilde{W}_{\text{small}}$ consists of all intervals I_h, which do no have enough room for processing all $\eta(X,h)$ jobs. Formally,

$$\widetilde{W}_{\text{big}} = \{I_h \in \widetilde{W} \mid \eta(X,h)\Delta_h \le \widetilde{\mu}^X(I_h,t)\}, \qquad \widetilde{W}_{\text{small}} = \{I_h \in \widetilde{W} \mid \eta(X,h)\Delta_h > \widetilde{\mu}^X(I_h,t)\}.$$

It is easy to construct a feasible s-t flow x in $\widetilde{G}_{\infty}^{X}$ such that

$$\begin{array}{rcll} x(s,j) & = & x(j,I_h) = 0, & \text{if } j \in H \setminus X, \\ x(j,I_h) & = & \widetilde{\mu}^X(j,I_h) = \Delta_h, & \text{if } j \in X \text{ and } I_h \in \Gamma(j) \cap \widetilde{W}_{\text{big}}, \\ x(I_h,t) & = & \begin{cases} \eta(X,h)\Delta_h, & \text{if } I_h \in \widetilde{W}_{\text{big}}, \\ \widetilde{\mu}^X(I_h,t), & \text{if } I_h \in \widetilde{W}_{\text{small}}. \end{cases} & \end{array}$$

The flow on the remaining arcs is defined in order to satisfy the conservation law. The value of this flow is equal to

$$\begin{aligned} \sum_{I_h \in \widetilde{W}} x(I_h,t) &= \sum_{I_h \in \widetilde{W}_{\text{big}}} x(I_h,t) + \sum_{I_h \in \widetilde{W}_{\text{small}}} x(I_h,t) \\ &= \sum_{I_h \in \widetilde{W}_{\text{big}}} \eta(X,h)\Delta_h + \sum_{I_h \in \widetilde{W}_{\text{small}}} \widetilde{\mu}^X(I_h,t). \end{aligned}$$

For $I_h \in \widetilde{W}_{\text{big}}$ characterized by $\eta(X,h)\Delta_h \le \widetilde{\mu}^X(I_h,t) = \min\{m-\eta(K,h), \eta(H,h)\}\Delta_h \le (m-\eta(K,h))\,\Delta_h$, we have

$$x(I_h,t) = \eta(X,h)\Delta_h = \min\{m-\eta(K,h), \eta(X,h)\}\Delta_h.$$

For $I_h \in \widetilde{W}_{\text{small}}$ characterized by $\widetilde{\mu}^X(I_h,t) < \eta(X,h)\Delta_h \le \eta(H,h)\Delta_h$, we have

$$x(I_h,t) = \widetilde{\mu}^X(I_h,t) = \min\{m-\eta(K,h), \eta(X,h)\}\Delta_h.$$

Thus, we deduce

$$\sum_{I_h \in \widetilde{W}} x(I_h,t) = \sum_{I_h \in \widetilde{W}} \min\{m-\eta(K,h), \eta(X,h)\}\Delta_h = \zeta.$$

Consider an s-t cut (S,T) given by

$$S = \{s\} \cup X \cup \widetilde{W}_{\text{small}}, \qquad T = \widetilde{V} \setminus S = \{t\} \cup (H \setminus X) \cup \widetilde{W}_{\text{big}},$$

and compute its capacity

$$\begin{aligned} \widetilde{\mu}^X(S,T) &= \sum_{j \in H \setminus X} \widetilde{\mu}^X(s,j) + \sum_{j \in X,\, I_h \in \Gamma(j) \cap \widetilde{W}_{\text{big}}} \widetilde{\mu}^X(j,I_h) + \sum_{I_h \in \widetilde{W}_{\text{small}}} \widetilde{\mu}^X(I_h,t) \\ &= 0 + \sum_{I_h \in \widetilde{W}_{\text{big}}} \eta(X,h)\Delta_h + \sum_{I_h \in \widetilde{W}_{\text{small}}} \widetilde{\mu}^X(I_h,t) = \sum_{I_h \in \widetilde{W}} x(I_h,t). \end{aligned}$$

Thus we have constructed the flow in $\widetilde{G}^X_\infty$ which is a maximum flow and it is of the required value ζ. ■

By Lemma 2, the problem (LP) in (26) to be solved in Step 2 can be reduced to the max-flow problem in network $\widetilde{G}_b$, which is obtained from network $\widetilde{G}_\infty(H,K)$ by replacing capacities $\widetilde{\mu}(u,v)$ with $\widetilde{\mu}_b(u,v)$ given by

$$\widetilde{\mu}_b(u,v) = \begin{cases} b(j), & \text{if } (u,v)=(s,j),\ j\in H,\\ \widetilde{\mu}(u,v), & \text{otherwise.}\end{cases}$$

For a maximum flow x_* in network $\widetilde{G}_b$, an optimal solution $\mathbf{q}\in\mathbb{R}^H$ to the problem (LP) is given by $q(j)=x_*(s,j)$, $j\in H$.

Now, in Step 3, we need to find a tight set Y_*, i.e., a set Y_* with $q(Y_*)=\psi(Y_*)$. The next lemma shows that a tight set can be obtained from a minimum s-t cut in $\widetilde{G}_b$. Thus, the problem to be solved in Step 3 is a min-cut problem in $\widetilde{G}_b$.

Lemma 3 *Let x_* and (S_*,T_*) be a maximum flow and a minimum s-t cut, respectively, in network $\widetilde{G}_b$. For vector $\mathbf{q}\in\mathbb{R}^H$ defined as $q(j)=x_*(s,j)$, $j\in H$, the equalities*

$$q(S_*\cap H)=\psi(S_*\cap H),\qquad q(j)=b(j),\ j\in H\setminus S_* \tag{28}$$

hold.

Proof: Since x_* is a maximum flow and (S_*,T_*) is a minimum s-t cut, Theorem 1 (i) implies that

$$\sum_{j\in H} x_*(s,j)=\widetilde{\mu}_b(S_*,T_*) \tag{29}$$

and $x_*(u,v)=\widetilde{\mu}_b(u,v)$ for every $(u,v)\in A(S_*,T_*)$. In particular, for each $j\in H\setminus S_*$, it holds that $(s,j)\in A(S_*,T_*)$, so that

$$x_*(s,j)=\widetilde{\mu}_b(s,j)=b(j),\quad j\in H\setminus S_*, \tag{30}$$

and the second equation in (28) holds.

For a minimum cut (S_*,T_*), define

$$X=H\cap S_*.$$

To verify the first equation in (28), we prove that

$$\sum_{j\in X} x_*(s,j)=\psi(X).$$

Let network $\widetilde{G}^X_\infty$ be as defined in the proof of Lemma 2. The lemma asserts that

$$\psi(X)=\max\left\{\sum_{j\in H} y(s,j)\ \middle|\ y \text{ is a feasible } s\text{-}t \text{ flow in } \widetilde{G}^X_\infty\right\}. \tag{31}$$

Let $x:\widetilde{A}\to\mathbb{R}$ be a feasible s-t flow in $\widetilde{G}^X_\infty$ such that

$$x(u,v)\le x_*(u,v),\ (u,v)\in\widetilde{A},\qquad x(s,j)=\begin{cases} x_*(s,j), & \text{if } j\in X,\\ 0, & \text{otherwise.}\end{cases}$$

Such a flow can be obtained easily from x_*; see, e.g., [1]. Notice that $H \setminus S_* = H \setminus (H \cap S_*) = H \setminus X$, so that (30) implies

$$\sum_{j \in H \setminus X} x_*(s,j) = \sum_{j \in H \setminus X} \widetilde{\mu}_b(s,j).$$

This and (29) yield

$$\sum_{j \in H} x(s,j) = \sum_{j \in H} x_*(s,j) - \sum_{j \in H \setminus X} x_*(s,j) = \widetilde{\mu}_b(S_*, T_*) - \sum_{j \in H \setminus X} \widetilde{\mu}_b(s,j) = \widetilde{\mu}^X(S_*, T_*), \quad (32)$$

where the last equality is deduced by comparing capacities $\widetilde{\mu}_b$ and $\widetilde{\mu}^X$ of arcs in the networks $\widetilde{G}_b$ and $\widetilde{G}^X_\infty$:

$$\begin{aligned}
&\text{for } (s,j) \in \widetilde{A} \text{ with } j \in X, && \widetilde{\mu}_b(s,j) = b(j),\ \widetilde{\mu}^X(s,j) = +\infty, \\
&\text{for } (s,j) \in \widetilde{A} \text{ with } j \in H \setminus X, && \widetilde{\mu}_b(s,j) = b(j),\ \widetilde{\mu}^X(s,j) = 0, \\
&\text{for other arcs } (u,v) \in \widetilde{A}, && \widetilde{\mu}_b(u,v) = \widetilde{\mu}^X(u,v).
\end{aligned}$$

It follows from (32) and Theorem 1 (ii) that x is a maximum flow in $\widetilde{G}^X_\infty$, which, together with (31), implies

$$\psi(X) = \sum_{j \in H} x(s,j) = \sum_{j \in X} x_*(s,j).$$

This concludes the proof. ∎

In summary, Steps 2 and 3 can be reduced to the max-flow problem and to the min-cut problem in network $\widetilde{G}_b$, respectively. Since $\widetilde{G}_b$ has $O(n)$ nodes, Step 2 requires $O(n^3)$ time, as mentioned in Section 2. Once we obtain a maximum flow, a minimum s-t cut in $\widetilde{G}_b$ can be found in $O(|\widetilde{A}|) = O(n^2)$ time; see, e.g., [1, 29]. Therefore, $T_{23}(n) = O(n^3)$ holds and the running time of Algorithm F-Decomp is $O(nT_2(n)) = O(n^4)$. Applying this algorithm, we find the actual processing times $p(j)$ of the jobs, and the optimal speeds are given as $s(j) = w(j)/p(j)$.

Theorem 2 *The SSP on m parallel machines to minimize the function (3) can be solved in $O(n^4)$ time.*

In the remainder of this section, we consider the SSP, assuming that the speed cost functions are job-independent, i.e., the speed cost function becomes Φ of the form (2). In terms of the decision variables $p(j)$, $j \in N$, the objective function in (17) is given as

$$\hat{\Phi} = \sum_{j=1}^{n} p(j)\ f\Big(\frac{w(j)}{p(j)}\Big), \quad (33)$$

where f is a convex function, common to all jobs.

We show that in this case, the problem can be solved faster. The basis of our reasoning is a non-trivial statement due to [26] and [27] that reduces this problem to a quadratic optimization problem.

Theorem 3 ([26, 27]) *The problem of minimizing the function (33) over a base polyhedron $B(\varphi)$ is equivalent to*

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n} \frac{p(j)^2}{w(j)} \\ \text{subject to} & \mathbf{p} \in B(\varphi), \end{array} \tag{34}$$

with a separable quadratic objective function.

Thus, to minimize function $\hat{\Phi}$ we do not need Algorithm F-Decomp. Instead, we can solve the problem (34) of minimizing a quadratic function over a base polyhedron. In terms of network flow, the latter problem is a problem of finding a flow in network G_∞ that minimizes a separable quadratic cost function, with non-zero costs only on the arcs going out of the source. Exactly such a problem is considered by Gallo et al. [13] and Hochbaum and Hong [16], who reduce it to the parametric max-flow problem and show how to solve it in $O\left(n^3\right)$ time. This observation can be summarized as the following statement.

Theorem 4 *The SSP on m parallel machines to minimize the function (2) can be solved in $O(n^3)$ time.*

Notice that the running time $O(n^3)$ established in Theorem 4 is several orders faster then the best previously known. For a more general problem in Theorem 2, with job-dependent speed costs, we are not aware of any prior results.

Remark 3 *The results described in this section for speed scaling problems on identical parallel machines compare favorably with the results on scheduling problems with controllable processing times for the same machine environment. For example, the problem of minimizing the total compression cost reduces to the min-cost max-flow problem with a linear objective cost function in a network similar to G_b. McCormick [23] shows that the latter problem can be reduced to the parametric max-flow problem and solved in $O(n^3)$ time.*

5 Solving SSP on a Single Machine

In this section, we explain the implementation details of Steps 2 and 3 of Algorithm F-Decomp in the case of a single machine, where we use a different approach from the one proposed in Section 4.

In Step 2, we need to solve the linear programming problem (LP) of type (26), where the function $\psi : 2^H \to \mathbb{R}$ is given by (21). Recall the explicit representation (25) of the function ψ:

$$\psi(X) = \sum_{I_h \in \Gamma(X \cup K) \setminus \Gamma(K)} \Delta_h, \quad X \subseteq H.$$

A meaningful interpretation of $\psi(X)$ is the total length of the time intervals originally available for processing the jobs of set $X \cup K$ after the intervals for processing the jobs of set K have been completely used up. Hence, problem (LP) corresponds to a single machine scheduling problem in which the jobs of set K have already been scheduled, and the jobs of set H must be scheduled in the remaining available time intervals. Denote $W_A = \Gamma(H \cup K) \setminus \Gamma(K) = \Gamma(H) \setminus \Gamma(K)$, which is the set of these available intervals, and assume that it consists of the intervals

$$[\tau_{\pi(1)-1}, \tau_{\pi(1)}], [\tau_{\pi(2)-1}, \tau_{\pi(2)}], \ldots, [\tau_{\pi(\nu)-1}, \tau_{\pi(\nu)}],$$

where $1 \leq \pi(1) < \pi(2) < \cdots < \pi(\nu)$. We see that the machine is busy (or unavailable) during the intervals

$$[\min_{j\in H} r(j), \tau_{\pi(1)-1}], [\tau_{\pi(1)}, \tau_{\pi(2)-1}], [\tau_{\pi(2)}, \tau_{\pi(3)-1}], \ldots, [\tau_{\pi(\nu-1)}, \tau_{\pi(\nu)-1}], [\tau_{\pi(\nu)}, \max_{j\in H} d(j)].$$

We denote the set of these busy intervals by W_B, i.e., $W_B = \Gamma(H) \cap \Gamma(K)$. In problem (LP), it is required to determine the actual processing times $q(j)$ of jobs of set H to maximize the total (unweighted) actual processing time $\sum_{j\in H} q(j)$, under the conditions that actual processing time $q(j)$ of each $j \in H$ satisfies $0 \leq q(j) \leq b(j)$, all jobs in H are scheduled within the ν intervals of set W_A, and no job $j \in H$ is scheduled outside the interval $[r(j), d(j)]$.

Problem (LP) is related to the problem of minimizing the total (unweighted) compression with zero lower bounds on actual processing times discussed in [17] and [36]. In particular, the algorithm by Hochbaum and Shamir [17] uses the UNION-FIND technique and solves the problem in $O(h+\nu)$ time under the assumption that jobs in H are appropriately sorted. The algorithm is based on the latest-release-date-first rule. Informally, the jobs are taken one by one in the order of their numbering and scheduled in a "backwards"manner: each job $j \in H$ is placed into the current partial schedule to fill the available time intervals consecutively, from right to left, starting from the right-most available interval. The assignment of a job j is complete either if its actual processing time $q(j)$ reaches its upper bound $b(j)$ or if no available interval within the interval $[r(j), d(j)]$ is left (recall that the intervals of set W_B are seen as busy).

For our purposes, however, we not only need the optimal values $q_*(j)$ of the processing times, but also a tight set, i.e., a set $Y_* \subseteq H$ with $q_*(Y_*) = \psi(Y_*)$. In scheduling terms, for Problem (LP) the jobs of a tight set completely use all intervals available for their processing. The actual processing time of a job in a tight set cannot be extended (even ignoring its upper bound) without compromising feasibility of the schedule. If a job is not in any tight set, then the job does not use the whole interval even if its processing time is fully extended (and could have been extended further if we had ignored the upper bound $b(j)$). Only a slight modification of the Hochbaum-Shamir algorithm, which does not affect its linear running time, leads to finding the required tight set Y_*. In the description of the algorithm, the jobs of set H are renumbered by the integers $1, 2, \ldots, h$ in non-increasing order of their release dates, i.e.,

$$r(1) \geq r(2) \geq \cdots \geq r(h); \tag{35}$$

additionally, if $r(j) = r(j+1)$ for some $j \in H$ then $d(j) \leq d(j+1)$ holds. For a schedule that is feasible for Problem (LP) under consideration, an interval during which the machine is permanently busy, possibly including the intervals from W_B, is called a *block*. Recall that a schedule delivered by the Hochbaum-Shamir algorithm can be seen as a collection of blocks separated by idle intervals.

Algorithm HSY

Step 1. Set $Y_*^0 := \emptyset$.

Step 2. For each job k from 1 to h do

(a) Schedule job k in accordance with the algorithm by [17].

(b) If in the current schedule the interval $[r(k), d(k)]$ has no idle time, then find a block B^k in which job k completes and determine the set Y^k of all jobs that complete in the same block; define $Y_*^k := Y_*^{k-1} \cup Y^k$. Otherwise (i.e., if in the current schedule the interval $[r(k), d(k)]$ has an idle time), define $Y_*^k := Y_*^{k-1}$.

Step 3. Output $Y_* := Y_*^h$ and stop.

In what follows we formulate the statements that show that Algorithm HSY finds the set Y_* correctly. Each lemma below is applied to schedule S_k, which is the schedule found in Step 2(a) for the jobs $1, \ldots, k$.

Lemma 4 *In schedule S_k any job $j \leq k$ starts and finishes in one block.*

Proof: Suppose $[t_1, t_2]$ and $[t_3, t_4]$, where $t_1 < t_2 < t_3 < t_4$, are two consecutive blocks in S_k such that job j, $j \leq k$, is processed in each of these blocks. Due to the feasibility of schedule S_k, we have $r(j) < t_2 < t_3 < d(j)$, i.e., the interval $[t_2, t_3]$ could be used for processing job j, but is left idle. This contradicts to the way the Hochbaum-Shamir algorithm operates. ∎

Lemma 5 *If the interval $[r(k), d(k)]$ has no idle time in schedule S_k, then Y^k is a tight set.*

Proof: Lemma 4 implies that in Step 2(b) of Algorithm HSY, Y^k is the set of jobs that start and complete in block B^k. Since job k has the smallest release date among all jobs in schedule S_k and the interval $[r(k), d(k)]$ has no idle time, it follows that block B^k is the interval $\hat{I} = [r(k), t]$, where $t = \max\left\{d(j) \mid j \in Y^k\right\}$. Let δ denote the total length of all intervals of set W_B within the interval $\hat{I}$. Then $q_*(Y^k) = t - r(k) - \delta$. On the other hand, no job of set Y^k can start before time $r(k)$, complete after time t and be assigned to the intervals of set W_B, so that $\psi(Y^k) = t - r(k) - \delta$. Thus, $q_*(Y^k) = \psi(Y^k)$. ∎

Notice that output Y_* in Step 3 is given as the union of sets $Y^1, Y^2, \ldots, Y^h$, and each Y^k $1 \leq k \leq h$) is a tight set by Lemma 5. Since the union of tight sets is again a tight set, the equality $q_*(Y_*) = \psi(Y_*)$ holds.

Recall that the Hochbaum-Shamir algorithm manipulates the intervals of machine availability organized in sets of contiguous intervals. In particular, it uses the FIND function to determine the set that contains any given original interval by retrieving the first interval in that set. Moreover, it uses the procedure UNION to merge two sets of intervals into a new set. Since the Hochbaum-Shamir algorithm actually determines the length of processing of each job k in the original intervals of availability, the required block B^k (the set of intervals that contains the latest interval for processing job k) will be found (see Step 2(b)). To be able to determine the set Y^k of the jobs in block B^k, we assume that for each block (or a set of intervals) the list of jobs assigned to be processed in this block is maintained. Once the jobs of set Y^k are added to set Y_*^k, the corresponding block together with its list of jobs is deleted. When two sets of intervals merge (a larger block is formed), the corresponding lists of jobs are linked. Thus, the running time of the original algorithm by Hochbaum and Shamir is not affected. Thus, we have proved that Algorithm HSY solves Problem (LP) and finds a tight set Y_* in $O(h + \nu)$ time.

Analyzing the overall time complexity of Algorithm F-Decomp we observe that the jobs can be renumbered in accordance with (35) once in $O(n \log n)$, and the relative order of the jobs does not change whenever some of the intervals are eliminated to produce a subproblem. As decomposition reduces to splitting the job set and the interval set, its time complexity is $O(h + \nu) = O(n)$, the same as the time complexity of Steps 2-3. Thus the overall running time of Algorithm F-Decomp is $O(n \log n + nT_{23}(n)) = O(n^2)$.

Theorem 5 *The SSP on a single machine to minimize the function (3) can be solved in* $O(n^2)$ *time.*

6 Conclusions

In our study, we have provided a new methodology for solving the SSP based on submodular optimization. Exploiting the properties of the underlying submodular optimization model for different versions of the SSP, we produce three efficient algorithms, two of which are based on the decomposition method by Fujishige [12].

For the model with a single machine and job-dependent speed cost functions f_j, the decomposition algorithm can be implemented in $O(n^2)$ time, outperforming the previous algorithms known for the special case with a single speed cost function f common for all jobs: the famous $O(n^3)$ YDS algorithm by [38] and the alternative $O(n^2 \log n)$ approach by [21]. Noteworthy, the YDS algorithm is recognized as the most cited result in the speed scaling literature. For a multi-machine model, our approach achieves a substantial speed up in comparison with the existing ones by [4] and [6].

The proposed new methodology provides a new insight into the underlying optimization model and demonstrates a potential of handling advanced features of enhanced models. It delivers the first efficient solution method for the most general multi-machine model with job-dependent speed cost functions f_j. Another enhancement we intend to consider in our future research is the case of non-identical machines. Unlike traditional research where processors have the same speed/energy characteristics, modern large scale computing environments, such as high performance computing systems, grids, clouds, server farms, etc., often deal with heterogeneous processors. It should be noted that theoretical research of the relevant multi-machine models is quite limited, as highlighted in the recent papers [2, 3].

Acknowledgement

This research was supported by the EPSRC funded project EP/J019755/1 "Submodular Optimisation Techniques for Scheduling with Controllable Parameters". The first author is partially supported by JSPS/MEXT KAKENHI Grand Numbers 24500002, 25106503.

Appendix: Connection Between YDS Algorithm and Fujishige's Monotone Algorithm

The correctness of the YDS algorithm for the speed scaling problem with the objective of the form (33) on a single machine [38] has been proved by Bansal et al. [10]. Their argument is based on the Karush-Kuhn-Tucker conditions. Below we discuss the connection of the YDS algorithm with the "monotone"algorithm by Fujishige [11] for minimizing a separable quadratic convex function of the form (34) over a base polyhedron.

A crucial concept for the YDS algorithm is the *density* of an interval I defined as the ratio of the processing volume of all jobs that can be started and completed within I to the length of interval I. The YDS algorithm can be seen as an iterative process. In each iteration, the algorithm finds an interval of maximum density and determines the set of jobs to be scheduled in I. In the corresponding partial schedule the identified jobs run within I at the speed equal to the established density and are preemptively scheduled in non-decreasing order of their deadlines (the EDF rule). In the next iteration, the algorithm deals with a

reduced instance of the problem, with interval I and the corresponding set of scheduled jobs removed.

A possible formal description of the YDS algorithm is given below. Let I_h, τ_h, and Δ_h be as defined in Section 2. In each iteration of the algorithm, K denotes the set of jobs which have already been scheduled, i.e., for each job $j \in K$, its processing time has been fixed. Also, W_A denotes the set of those intervals I_h, $1 \leq h \leq \gamma$, in which the machine is idle, with no job assigned yet.

Algorithm YDS

Step 0. Set

$$K := \emptyset, \qquad W_A := \{I_h \mid h = 1, 2, \ldots, \gamma\}.$$

Step 1. Considering the jobs of set $N \setminus K$, find the idle interval of maximum density and the set of jobs Y_* to be scheduled in that interval. For these purposes, determine the values $\tau', \tau'' \in \{\tau_0, \tau_1, \ldots, \tau_\gamma\}$ with $\tau' < \tau''$ that maximize the ratio

$$s = \frac{\sum\limits_{j \in N \setminus K,\, [r(j), d(j)] \subseteq [\tau', \tau'']} w(j)}{\sum\limits_{I_h \in W_A,\, I_h \subseteq [\tau', \tau'']} \Delta_h},$$

where the numerator represents the total processing volume of the unscheduled jobs of set $N \setminus K$ that can be scheduled in the time interval $[\tau', \tau'']$, while the denominator, which must be positive, is equal to the total length of the idle intervals that belong to $[\tau', \tau'']$. Define

$$Y_* = \{j \in N \setminus K \mid [r(j), d(j)] \subseteq [\tau', \tau'']\}.$$

Step 2. Set $p_*(j) = (1/s)w(j)$ for each $j \in Y_*$.

Step 3. Update

$$K := K \cup Y_*, \qquad W_A := W_A \setminus \{I_h \in W_A \mid I_h \subseteq [\tau', \tau'']\}.$$

If $K = N$, then output the vector $\mathbf{p}_* = (p_*(j) \mid j \in N)$ and stop; otherwise, go to Step 1.

The algorithm outputs the found actual processing time for each job. The corresponding schedule can be found by applying the EDF rule to schedule the jobs of set Y_* in the interval $I = [\tau', \tau'']$. Our description of Algorithm YDS is slightly different from its traditional presentation, see, e.g., [2, 38], where in each iteration to prevent the assignment of unscheduled jobs to already occupied intervals their release dates and/or deadlines are appropriately updated. The version of the algorithm given above achieves this by keeping track of set of W_A of idle intervals.

In each iteration of Algorithm YDS, it can be shown that if Step 1 outputs the maximum density s achieved for some interval and the jobs of set Y_*, then the same density is also achieved for the interval $[\tau', \tau'']$, where

$$\tau' = \min\{r(j) \,|\, j \in Y_*\}, \ \tau'' = \max\{d(j) \,|\, j \in Y_*\}.$$

This allows us to reformulate Step 1 of Algorithm YDS in set-selecting terms:

Step 1′ Find a nonempty set $Y_* \subseteq N \setminus K$ that maximizes the value

$$s = \frac{w(Y_*)}{\sum_{I_h \in W_A,\, I_h \subseteq [\tau', \tau'']} \Delta_h},$$

where $\tau' = \min\{r(j) \mid j \in Y_*\}$ and $\tau'' = \max\{d(j) \mid j \in Y_*\}$.

Notice that set Y_* selected in Step 1′ satisfies $\sum_{I_h \in \mathcal{I},\, I_h \subseteq [\tau', \tau'']} \Delta_h > 0$; this follows from the choice of Y_* in each iteration.

We can show by induction on the number of iterations that $W_A = \Gamma(N) \setminus \Gamma(K)$ holds at the beginning of each iteration in Algorithm YDS with Step 1′ instead of Step 1. Moreover, the set of intervals $\{I_h \in W_A \mid I_h \subseteq [\tau', \tau'']\}$ involved in computation of s in Step 1′ can be rewritten as $\Gamma(Y_* \cup K) \setminus \Gamma(K)$, i.e.,

$$s = \frac{w(Y_*)}{\sum_{I_h \in \Gamma(Y_* \cup K) \setminus \Gamma(K)} \Delta_h} = \frac{w(Y_*)}{\varphi(Y_* \cup K) - \varphi(K)}, \tag{36}$$

where the last equality is by the explicit representation (25) of ψ. It should be noted that for set Y_* selected in Step 1′ the union of intervals $[r(j), d(j)]$ for $j \in Y_*$ may yield several disjoint intervals. In such a case, we can easily show that each of the disjoint intervals has the maximum density among all possible intervals.

In summary, Algorithm YDS can be rewritten as follows.

Algorithm YDS-Mono

Step 0. Set $K := \emptyset$.

Step 1. Compute

$$s = \max\left\{ \frac{w(X)}{\varphi(X \cup K) - \varphi(K)} \;\middle|\; \emptyset \neq X \subseteq N \setminus K \right\}.$$

Find a nonempty set $Y_* \subseteq N \setminus K$ satisfying $w(Y_*)/(\varphi(Y_* \cup K) - \varphi(K)) = s$.

Step 2. Set $p_*(j) = (1/s)w(j)$ for each $j \in Y_*$.

Step 3. Set $K := K \cup Y_*$. If $K = N$, then output the vector $\mathbf{p}_* = (p_*(j) \mid j \in N)$ and stop; otherwise, go to Step 1.

As follows from Section 3, in terms of optimization with submodular constraints, the SSP with a single machine and job-dependent speed cost functions can be formulated as (17), where $f_j = f$, $j \in N$, and the rank function $\varphi(X)$, $X \subseteq N$, is given by (15). Due to Theorem 3, the problem can be reduced to a quadratic optimization problem (34), which is known to be solvable by a so-called monotone algorithm that is described as follows. See [11] and [12, Section 9.2] for more details on this algorithm.

Algorithm F-Mono

Step 0. Set $K := \emptyset$.

Step 1. Compute

$$\begin{aligned} \lambda_* &= \max\{\lambda \mid \lambda w(X) \leq \varphi(X \cup K) - \varphi(K)\ (\forall X \subseteq N \setminus K)\} \\ &= \min\left\{ \frac{\varphi(X \cup K) - \varphi(K)}{w(X)} \;\middle|\; \emptyset \neq X \subseteq N \setminus K \right\}. \end{aligned}$$

Find a nonempty set $Y_* \subseteq N \setminus K$ satisfying $\lambda_* w(Y_*) = \varphi(Y_* \cup K) - \varphi(K)$.

Step 2. Set $p_*(j) := \lambda_* w(j)$ for each $j \in Y_*$.

Step 3. Set $K := K \cup Y_*$. If $K = N$, then output the vector $\mathbf{p}_* = (p_*(j) \mid j \in N)$ and stop. Otherwise, go to Step 1.

Remark 4 *In the original version of the monotone algorithm in [11] and [12, Section 9.2], the set Y_* in Step 1 is a maximal set with $\lambda_* w(Y_*) = \varphi(Y_* \cup K) - \varphi(K)$. This maximality condition, however, can be removed, and the algorithm still works; see also Remark 1.*

Algorithm F-Mono can be seen as a specialized implementation of Algorithm F-Decomp presented in Section 3 for a separable quadratic convex objective function. Indeed, Algorithm F-Decomp is developed in [15] to make the original monotone algorithm from [11] able to minimize a general separable convex function over a base polyhedron.

Coming back to the speed scaling problem on a single machine observe that a nonempty set $Y_* \subseteq N \setminus K$ maximizes the ratio $w(Y_*)/\left(\varphi(Y_* \cup K) - \varphi(K)\right)$ if and only if it minimizes the ratio $\left(\varphi(Y_* \cup K) - \varphi(K)\right)/w(Y_*)$. This means that the maximum density s is equal to $1/\lambda_*$, where λ_* is defined as in Step 1 of Algorithm F-Mono. Hence, Algorithm YDS-Mono is nothing but the monotone algorithm. That is, Algorithm YDS can be seen as an implementation of the monotone algorithm specialized for the speed scaling problem on a single machine.

Notice that the conjecture on a possible improvement of Algorithm YDS was claimed by its authors in 1995, but remained unsupported for more than 10 years, when a faster algorithm was developed in [21]. Eventually, an $O\left(n^2\right)$-time implementation is given in [22]. All these improved algorithms remain versions of Algorithm F-Decomp.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications,* Prentice Hall, Englewood Cliffs, 1993.

[2] S. Albers, *Algorithms for energy saving*, in Efficient Algorithms, S. Albers, H. Alt, and S. Näher, eds., LNCS 5760, Springer, Berlin, 2009, pp. 173–186.

[3] S. Albers, *Energy-efficient algorithms*, Comm. ACM, 53 (2010), pp. 86–96.

[4] S. Albers, A. Antoniadis and G. Geiner, *On multiprocessor speed scaling with migration*, J. Comput. System Sci., 81 (2015), pp. 1194–1209.

[5] S. Albers, F. Müller and S. Schmelzer, *Speed scaling on parallel processors*, Algorithmica, 68 (2007), pp. 404–425.

[6] E. Angel, E. Bampis, F. Kacem and D. Letsios, *Speed scaling on parallel processors with migration*, in Proceedings of Euro-Par 2012, LNCS 7484 (2012), pp. 128–140.

[7] M. Bambagini, G. Buttazzo and M. Bertogna, *Energy-aware scheduling for tasks with mixed energy requirements*, in Proceedings of 4th International Real-Time Scheduling Open Problems Seminar, Paris, France, 2013.

[8] M. Bambagini, J. Lelli, G. Buttazzo and G. Lipari, *On the energy-aware partitioning of real-time tasks on homogeneous multi-processor systems*, in Proceedings of the 4th International Conference on Energy Aware Computing, Istanbul, Turkey, 2013, pp. 69–74.

[9] E. Bampis, D. Letsios and G. Lucarelli. *Green scheduling, flows and matchings,* Theor. Comp. Sci., 579 (2015), 126–136.

[10] N. Bansal, T. Kimbrel and K. Pruhs. *Speed scaling to manage energy and temperature,* J. ACM, 54 (2007), No 1, Article 3.

[11] S. Fujishige, *Lexicographically optimal base of a polymatroid with respect to a weight vector*, Math. Oper. Res., 5 (1980), pp. 186–196.

[12] S. Fujishige, *Submodular Functions and Optimization*, 2nd ed., Ann. Discrete Math., 58, Elsevier, Amsterdam, 2005.

[13] G. Gallo, M. D. Grigoriadis and R. E. Tarjan, *A fast parametric maximum flow algorithm and applications*, SIAM J. Comput., 18 (1989), pp. 30–55.

[14] V. S. Gordon and V. S. Tanaev, *Deadlines in single-stage deterministic scheduling*, in Optimization of Systems for Collecting, Transfer and Processing of Analogous and Discrete Data in Local Information Computing Systems, Materials of the 1st Joint Soviet-Bulgarian seminar (Institute of Engineering Cybernetics of Academy of Sciences of BSSR – Institute of Engineering Cybernetics of Bulgarian Academy of Sciences, Minsk), 1973, pp. 53–58 (in Russian).

[15] H. Groenevelt, *Two algorithms for maximizing a separable concave function over a polymatroid feasible region*, European J. Oper. Res., 54 (1991), pp. 227–236.

[16] D. S. Hochbaum and S.-P. Hong, *About strongly polynomial time algorithms for quadratic optimization over submodular constraints*, Math. Program.,69 (1995), pp. 269–309.

[17] D. S. Hochbaum and R. Shamir, *Minimizing the number of tardy job unit under release time constraints.* Discrete Appl. Math., 28 (1990), pp. 45–57.

[18] W. Horn, *Some simple scheduling algorithms*, Naval Res. Logist. Quart., 21 (1974), pp. 177–185.

[19] H. Ishii, C. Martel, T. Masuda and T. Nishida, *A generalized uniform processor system*, Oper. Res., 33 (1985), pp. 346–362.

[20] A. V. Karzanov, *Determining the maximal flow in an network by the method of preflows*, Soviet Math. Dokl., 15 (1974), pp. 434–437.

[21] M. Li, A. C. Yao and F. F. Yao, *Discrete and continuous min-energy schedules for variable voltage processors*, in Proc. Nat. Acad. Sci. USA, 103 (2006), pp. 3983–3987.

[22] M. Li, F. F. Yao and H. Yuan, *An $O(n^2)$ algorithm for computing optimal continuous voltage schedules*, `arxiv:1408.5995v1` (2014).

[23] S. T. McCormick, *Fast algorithms for parametric scheduling come from extensions to parametric maximum flow*, Oper. Res., 47 (1999), pp. 744–756.

[24] R. McNaughton, *Scheduling with deadlines and loss functions*, Manag. Sci., 12 (1959), pp. 1–12.

[25] N. Megiddo, *Optimal flows in networks with multiple sources and sinks*, Math. Program., 7 (1974), pp. 97–107.

[26] K. Murota, *Note on the universal bases of a pair of polymatroids*, J. Oper. Res. Soc. Japan, 31 (1988), pp. 565–573.

[27] K. Nagano and K. Aihara, *Equivalence of convex minimization problems over base polytopes*, Japan J. Indust. Appl. Math., 29 (2012), pp. 519–534.

[28] E. Nowicki and S. Zdrzalka, *A survey of results for sequencing problems with controllable processing times*, Discrete Appl. Math., 26 (1990), pp. 271–287.

[29] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, Berlin, 2003.

[30] D. Shabtay and G. Steiner, *A survey of scheduling with controllable processing times*, Discrete Appl. Math.,155 (2007), pp. 1643–1666.

[31] N. V. Shakhlevich and V. A. Strusevich, *Pre-emptive scheduling problems with controllable processing times*, J. Sched., 8 (2005), pp. 233–253.

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

[33] N. V. Shakhlevich, A. Shioura and V. A. Strusevich, *Single machine scheduling with controllable processing times by submodular optimization*, Int. J. Found. Comput. Sci., 20 (2009), pp. 247–269.

[34] A. Shioura, N. V. Shakhlevich and V. A. Strusevich, *A submodular optimization approach to bicriteria scheduling problems with controllable processing times on parallel machines*, SIAM J. Discrete. Math., 27 (2013), pp. 186–204.

[35] A. Shioura, N. V. Shakhlevich and V. A. Strusevich, *Decomposition algorithms for submodular optimization with applications to parallel machine scheduling with controllable processing times*, Math. Program, In Press, doi:10.1007/s10107-014-0814-9

[36] W.-K. Shih, J. W. S. Liu and J.-Y. Chung, *Algorithms for scheduling imprecise computations with timing constraints*, SIAM J. Comput., 20 (1991), pp. 537–552.

[37] V. Venkatachalam and M. Franz, *Power reduction techniques for microprocessor systems*, ACM Computing Surveys, 37 (2005), pp. 195–237.

[38] F. F. Yao, A. J. Demers and S. Shenker, *A scheduling model for reduced CPU energy*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 374–382.