

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

Akiyoshi Shioura,^a Natalia V. Shakhlevich,^b Vitaly A. Strusevich^c

^aDepartment of Industrial Engineering and Economics, Tokyo Institute of Technology, Tokyo 152, Japan; ^bSchool of Computing, University of Leeds, Leeds LS2 9JT, United Kingdom; ^cDepartment of Mathematical Sciences, University of Greenwich, Old Royal Naval College, London SE10 9LS, United Kingdom

Contact: shioura.a.aa@m.titech.ac.jp (AS); n.shakhlevich@leeds.ac.uk (NVS); v.strusevich@greenwich.ac.uk,

 <http://orcid.org/0000-0002-4602-8573> (VAS)

Received: September 09, 2015

Revised: January 16, 2017

Accepted: February 14, 2017


Published Online: September 28, 2017

<https://doi.org/10.1287/ijoc.2017.0758>

Copyright: © 2017 The Author(s)

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. Additionally, it efficiently handles the most general models with job-dependent speed/energy functions with single and multiple machines. To the best of our knowledge, this has 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.

History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms.

 **Open Access Statement:** This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as “*INFORMS Journal on Computing*”. Copyright © 2017 The Author(s). <https://doi.org/10.1287/ijoc.2017.0758>, used under a Creative Commons Attribution License: <https://creativecommons.org/licenses/by/4.0/>.

Funding: This research was supported by the EPSRC funded project “Submodular Optimisation Techniques for Scheduling with Controllable Parameters” [EP/J019755/1]. The first author is partially supported by JSPS/MEXT KAKENHI [Grants 24500002, 25106503].

Supplemental Material: The online appendix is available at <https://doi.org/10.1287/ijoc.2017.0758>.

Keywords: analysis of algorithms • computational complexity • programming • nonlinear • production-scheduling: single machine • production-scheduling: multiple machines

1. Introduction

Scheduling models with variable machine speeds have been studied since the 1980s; see, e.g., Ishii et al. (1985). They were reintroduced in the 1990s in the context of energy efficiency of battery operated portable computing devices; see Yao et al. (1995). 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 multiprocessor 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 paper by Yao et al. (1995) provides a fundamental algorithmic technique

widely known as YDS algorithm, 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. See Online Supplement 1 for a description of that algorithm and discussions, including its faster implementations. Notice that the same model and a very similar algorithm were also presented in an earlier paper by Vizing et al. (1981).

The multiprocessor version of the problem received attention quite recently, see Albers et al. (2007, 2015), Angel et al. (2012). The fastest strongly polynomial-time algorithm proposed in Albers et al. (2015) solves repeatedly a series of the maximum 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 leads to an $O(n^4 \log n)$ -time algorithm. The link to the maximum flow problem is also exploited in Angel et al. (2012), although 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- and multiprocessors 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. Although it is traditionally assumed that the energy consumption functions are identical for all jobs, in reality heterogeneous jobs may differ in their energy characteristics (e.g., as a result of their different read/write characteristics, the sizes of input/output files, the usage of internal and external memory, etc.). We demonstrate that the more general SSP with job-dependent energy consumption functions can be solved by the submodular optimization techniques in $O(n^2)$ and $O(n^4)$ time for the single- and multimachine 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.

The need to consider individual energy models for tasks dependent on their computation intensity or data intensity is widely recognized in the computing community, and more realistic features of the effect of speed scaling have been included in the models. Increasing processor's speed can speed up the computation part of the job, keeping the overheads of read/write operations unchanged. Therefore, the energy consumption function depends on job characteristics related to the job splitting into a computation part and an input/output part; see, e.g., Venkatachalam and Franz (2005) for an overview of the models of this type discussed in the context of memory bounded applications, Bambagini et al. (2013a, b) for an example of a job-dependent energy function, and Wu et al. (2012) for an example of mathematical analysis of the relevant model.

Our study of the problems with *heterogeneous jobs* complements another stream of research with *heterogeneous machines*. While in the models we consider the energy consumption functions are *job-dependent* and they can be different for different jobs even if the jobs are assigned to the same machine, in the models with *heterogeneous machines* the energy consumption functions are *machine dependent*. This means that the same job, when assigned to different machines, incurs different costs even if it is processed at the same speed. These type of problems are studied by Gupta et al. (2010, 2012), Bampis et al. (2016), Albers et al. (2016).

We now formally define the SSP with *heterogeneous jobs*. There is given a set of jobs $N = \{1, 2, \dots, n\}$ that have to be processed either on a single machine M_1 or on parallel machines M_1, M_2, \dots, M_m , where $m \geq 2$.

Each job $j \in N$ is given a *release date* $r(j)$, before which it is not available, 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 that 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 nondecreasing 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. As a result of the convexity of f , the energy can be minimized by processing each job j with a fixed speed $s(j)$, which does not change during the whole processing of a job; see, e.g., Albers et al. (2015). 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 nondecreasing.

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

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

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

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

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 the surveys by Nowicki and Zdrzałka (1990), Shabtay and Steiner (2007). To demonstrate the link between the problems with controllable processing times and the SSP, 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 $N = \{1, 2, \dots, n\}$ have to be processed with preemption either on a single machine M_1 or on parallel machines M_1, M_2, \dots, M_m , where $m \geq 2$. Each job j has a release date $r(j)$ and a deadline $d(j)$. A decision needs to be made about the actual duration $p(j)$ of a job: it should belong to a given interval $[l(j), u(j)]$. Such a decision results in *compression* of the longest processing time $u(j)$ down to $p(j)$, and the value $z(j) = u(j) - p(j)$ 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 Hochbaum and Shamir 1990 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 (Shakhlevich and Strusevich 2005, 2008; Shakhlevich et al. 2009; Shioura et al. 2013, 2015, 2016). 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 nonlinear 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 minimum-cost maximum-flow problem with a nonlinear convex separable objective function; see Section 2. The latter problem is then linked to a nonlinear convex minimization problem under submodular constraints, which can be solved by adapting a decomposition algorithm of Fujishige (1980); 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 $O(n^4)$ time on parallel machines and in $O(n^2)$ time on a single machine. In the multimachine case with the objective function Φ of the form (2) we rely on a nontrivial result in Murota (1988) and Nagano and Aihara (2012) to reduce the problem to the minimization problem with a separable quadratic objective, which allows the SSP to be solved in $O(n^3)$ time.

2. Reduction of Speed Scaling Problems to Minimum-Cost Flow Problems

Given a set $N = \{1, 2, \dots, n\}$ of jobs to be processed on either a single machine M_1 or on m parallel machines M_1, M_2, \dots, 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 F of the form (1). 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 $j \in N$. Then, the objective function F becomes

$$\hat{F} = \sum_{j=1}^n p(j) f_j \left(\frac{w(j)}{p(j)} \right). \quad (3)$$

This function has to be minimized over all feasible values of $p(j)$. We reformulate the resulting problem as a minimum-cost maximum-flow problem in a bipartite network with a nonlinear convex objective.

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 $\tau_0, \tau_1, \dots, \tau_\gamma$, 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 $W = \{I_h \mid 1 \leq h \leq \gamma\}$. 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 $\Gamma(j)$, i.e.,

$$\Gamma(j) = \{I_h \in W \mid I_h \subseteq [r(j), d(j)]\}. \quad (4)$$

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

$$\Gamma(X) = \bigcup_{j \in X} \Gamma(j). \quad (5)$$

Introduce the following bipartite network $G_\infty = (V, A)$ (see Figure 1 for an illustration). The node set is given by $V = \{s, t\} \cup N \cup W$, where s is the source node, t is the sink node, N is the set of job nodes, and W is the set of interval nodes, i.e., $W = \{I_1, I_2, \dots, I_\gamma\}$. The arc set A is given as $A = A^s \cup A^0 \cup A^t$, where

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

so that the source node s is connected to each job node, each interval node is connected to the sink node t , and each job node is connected to the nodes associated with the available intervals. We define the arc capacity $\mu: A \rightarrow \mathbb{R}_+$ as follows:

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

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. Gordon and Tanaev 1973, Horn 1974). *Let $\mathbf{p} = (p(1), \dots, p(n))$ 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 $x: A \rightarrow \mathbb{R}_+$ in network G_∞ with $x(s, j) = p(j)$ for all $j \in N$.*

For a network with a set of nodes V , an algorithm developed by Karzanov (1974) 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 (1959) 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 (3), 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 convex function with respect to $x(s, j)$. The cost of flow on each remaining arc 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 minimum-cost maximum-flow problem has a separable convex objective function. A similar formulation can be found in several papers on the SSP; see, e.g., Bampis et al. (2015) 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 Section 3.

3. Links to Submodular Optimization

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 by Fujishige (2005) and Schrijver (2003).

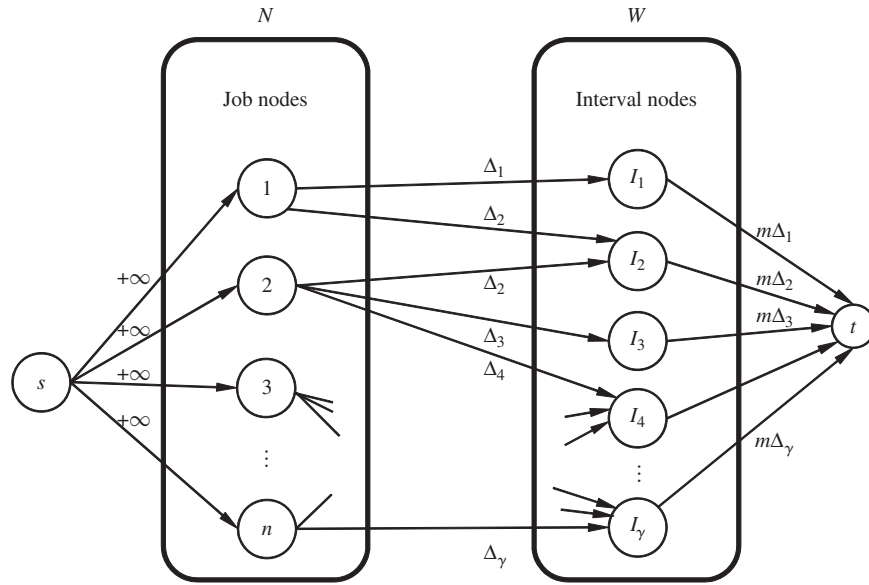
A set function $\varphi: 2^N \rightarrow \mathbb{R}$ is called *submodular* if the inequality

$$\varphi(X \cup Y) + \varphi(X \cap Y) \leq \varphi(X) + \varphi(Y) \quad (6)$$

holds for all sets $X, Y \in 2^N$, and called *monotone* if $\varphi(X) \leq \varphi(Y)$ for every $X, Y \in 2^N$ with $X \subseteq Y$. For a monotone submodular function φ defined on 2^N such that $\varphi(\emptyset) = 0$, the pair (N, φ) is called a *polymatroid* and φ the *rank function* of the polymatroid. For a polymatroid (N, φ) , define two polyhedra

$$\begin{aligned} P_{(+)}(\varphi) &= \{\mathbf{p} \in \mathbb{R}^N \mid p(X) \leq \varphi(X), X \in 2^N, \mathbf{p} \geq 0\}, \\ B(\varphi) &= \{\mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P_{(+)}(\varphi), p(N) = \varphi(N)\}, \end{aligned} \quad (7)$$

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



where we denote $p(X) = \sum_{j \in X} p(j)$ for a vector $\mathbf{p} \in \mathbb{R}^N$ and a set $X \subseteq N$. Polyhedra $P_{(+)}(\varphi)$ and $B(\varphi)$ are called a *polymatroid polyhedron* and a *base polyhedron*, respectively, associated with the polymatroid. Notice that $B(\varphi)$ represents the set of all maximal vectors in $P_{(+)}(\varphi)$.

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

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

It is known that such a polyhedron is a polymatroid polyhedron (see, e.g., Megiddo 1974, Lemma 4.1; Fujishige 2005, Section 2.2; Hochbaum and Hong 1995). Furthermore, all possible maximum flows can be characterized as a base polyhedron $B(\varphi)$ with a polymatroid rank function $\varphi: 2^N \rightarrow \mathbb{R}$ given by

$$\varphi(X) = \max \left\{ \sum_{j \in X} y(s, j) \mid y \text{ is a feasible } s\text{-}t \text{ flow in } G_\infty \right\}, \quad X \subseteq N. \quad (8)$$

Note that the polymatroid rank function φ can also be represented as

$$\varphi(X) = \max \left\{ \sum_{j \in N} y(s, j) \mid y \text{ is a feasible } s\text{-}t \text{ flow in } G_\infty^X \right\}, \quad X \subseteq N, \quad (9)$$

where G_∞^X is a network that 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.

We see from (8) and from the definition of the network G_∞ that the value $\varphi(X)$ admits an explicit formula as follows. For $X \subseteq N$ and $h = 1, 2, \dots, \gamma$, we denote by $\eta(X, h)$ the number of jobs in X that can be processed in the interval I_h , i.e.,

$$\eta(X, h) = |\{j \in X \mid I_h \in \Gamma(j)\}|. \quad (10)$$

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

$$\eta(X \cup Y, h) = \eta(X, h) + \eta(Y, h). \quad (11)$$

Then, the value $\varphi(X)$ is explicitly given as

$$\varphi(X) = \sum_{h=1}^{\gamma} \min\{m, \eta(X, h)\} \cdot \Delta_h, \quad X \subseteq N. \quad (12)$$

In scheduling terms, the value in the right-hand side of (12) 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. For the case of a single machine, i.e., for $m = 1$, the formula (12) can be simplified as

$$\varphi(X) = \sum_{I_h \in \Gamma(X)} \Delta_h, \quad X \subseteq N, \quad (13)$$

where $\Gamma(X)$ is given by (5).

Recall that in Section 2, the SSP is reduced to the minimum-cost maximum-flow problem in network G_∞ . Thus, in terms of submodular optimization, the SSP can be reformulated as

$$\begin{aligned} \text{SSP:} \quad & \text{minimize} \quad \sum_{j=1}^n p(j) f_j \left(\frac{w(j)}{p(j)} \right) \\ & \text{subject to} \quad \mathbf{p} \in B(\varphi), \end{aligned} \quad (14)$$

Downloaded from informs.org by [193.60.78.43] on 20 November 2017, at 02:37. For personal use only, all rights reserved.

with the polymatroid rank function φ defined by (8) with respect to network G_∞ . The problem (14) falls into the category of problems of minimizing convex separable functions with a polymatroid constraint:

$$\begin{aligned} & \text{minimize} && \sum_{j=1}^n h_j(p(j)) \\ & \text{subject to} && \mathbf{p} \in B(\varphi), \end{aligned} \quad (15)$$

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 (15) coincides with problem (14). Recall that every $\mathbf{p} \in B(\varphi)$ is a nonnegative vector since φ is a polymatroid rank function.

To solve the problem (15), we can adapt a decomposition algorithm by Groenevelt (1991) (see also Fujishige 2005, Section 8.2). A description of a variant of the decomposition algorithm that suits our purposes is given next.

Algorithm F-Decomp

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

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

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

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

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

$$q(Y_*) = \varphi(Y_*); \quad q(j) = b(j), \quad j \in N \setminus Y_*. \quad (16)$$

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:

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

Step 6. Find an optimal solution $\mathbf{p}_2 \in \mathbb{R}^{N \setminus Y_*}$ of the following problem:

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

Step 7. Output the direct sum $\mathbf{p}^* = \mathbf{p}_1 \oplus \mathbf{p}_2 \in \mathbb{R}^N$ and stop.

The decomposition algorithm 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 some jobs. The required feasible values $q(j)$, $j \in N$, are found in Step 2. The 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 because of the insufficient processing capacity, while for each of the remaining jobs in $N \setminus Y_*$ its actual duration can be further extended.

Notice that the subproblems to be solved in Steps 5 and 6 share a common structure. Using two sets $H, K \subseteq N$ such that $H \cap K = \emptyset$, each subproblem can be written in the following form:

$$\begin{aligned} & \text{minimize} && \sum_{j \in H} h_j(p(j)) \\ & \text{subject to} && p(X) \leq \psi_K(X), \quad X \in 2^H, \\ & && p(H) = \psi_K(H), \end{aligned} \quad (17)$$

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

$$\psi_K(X) = \varphi(X \cup K) - \varphi(K), \quad X \in 2^H. \quad (18)$$

Hence, the original problem can be solved recursively.

It should be noted that the function ψ_K is dependent on the set K , i.e., for $X \subseteq H$, the value $\psi_K(X)$ is different for different K in general. In the following, we omit the subscript K of ψ_K since it is clear from the context.

We now present the explicit representation of the function $\psi = \psi_K$ for problem (17). Substituting the expression (12) of φ into (18) we obtain

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

For $h = 1, 2, \dots, \gamma$, if $I_h \notin \Gamma(X)$ then we have $\eta(X \cup K, h) = \eta(K, h)$ and

$$\min\{m, \eta(X \cup K, h)\} = \min\{m, \eta(K, h)\}. \quad (19)$$

Also, if $I_h \in \Gamma(X)$ but $\eta(K, h) \geq m$, then again (19) holds. In either case, the corresponding term does not contribute to $\psi(X)$.

Thus, we only need to consider intervals of the set

$$W(X, K) = \{I_h \mid I_h \in \Gamma(X), \eta(K, h) < m\}, \quad (20)$$

namely those intervals I_h , which are suitable for processing the jobs from X (i.e., $I_h \in \Gamma(X)$) and are not fully used by the jobs from K (i.e., $\eta(K, h) < m$). 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 a result of (11). Thus, we have

$$\psi(X) = \sum_{I_h \in W(X, K)} \min\{m - \eta(K, h), \eta(X, h)\} \Delta_h. \quad (21)$$

In particular, if $m = 1$ then $W(X, K) = \Gamma(X) \setminus \Gamma(K)$ and

$$\psi(X) = \sum_{I_h \in \Gamma(X) \setminus \Gamma(K)} \Delta_h = \sum_{I_h \in \Gamma(X \cup K) \setminus \Gamma(K)} \Delta_h. \quad (22)$$

Remark 1. Conditions (16) used in Step 3 consist of a so-called *tightness* condition $q(Y_*) = \varphi(Y_*)$ for the set Y_* and the requirement that $q(j) = b(j)$ for $j \in N \setminus Y_*$. In the version of the algorithm presented by Fujishige (2005), the author recommends that Y_* is a maximal (and, therefore, unique) tight set with respect to the vector \mathbf{q} found in Step 2, but the maximality of the tight set is stated as not crucial.

Our version of the decomposition algorithm uses any tight set Y_* (not necessarily a maximal one), if it satisfies the second condition in (16). Such a set is called *instrumental set* in Shioura et al. (2015). We describe how to find an instrumental set Y_* for our two scheduling applications in Sections 4 and 5.

Notice that the maximal tight set Y_* recommended by Fujishige (2005) satisfies both conditions in (16) and therefore it is an instrumental set as well.

Remark 2. Step 1 involves minimization of a non-linear function. 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- 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 $f_j(s(j)) = s(j)^3$.

In Sections 4 and 5, 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, respectively. In fact, we only need to focus on Steps 2 and 3 that arise when solving the subproblem (17). Step 2 can be interpreted as finding an optimal solution q_* to the following auxiliary linear programming problem:

$$\begin{aligned} \text{(LP):} \quad & \text{maximize} \quad \sum_{j \in H} q(j) \\ & \text{subject to} \quad q(X) \leq \psi(X), \quad X \in 2^H, \\ & \quad \quad \quad 0 \leq q(j) \leq b(j), \quad j \in H, \end{aligned} \quad (23)$$

while Step 3 requires finding a set Y_* satisfying $q(Y_*) = \varphi(Y_*)$ and $q(j) = b(j)$, $j \in N \setminus Y_*$.

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 (17), where function $\psi: 2^H \rightarrow \mathbb{R}$ is given by (18) and the speed cost functions f_j are job dependent. We show that for solving (17), 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 (8) for function φ .

Let $\tilde{G}_\infty(H, K) = (\tilde{V}, \tilde{A})$ be a subgraph of G_∞ induced by the set of nodes $\tilde{V} = \{s, t\} \cup H \cup \tilde{W}$, where $\tilde{W} = W(H, K)$ is defined in accordance with (20). Thus, network $\tilde{G}_\infty(H, K)$ contains the job nodes associated with the jobs of set H and the interval nodes associated with intervals of set $\tilde{W} = W(H, K)$, 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 $\tilde{G}_\infty(H, K)$ is given as

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

and the arc capacities are as follows:

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

Note that $\tilde{\mu}(I_h, t)$ are defined in accordance with (21).

Lemma 2 shows that for any $X \subseteq H$ the corresponding value of $\psi(X)$ is defined as a maximum flow value in $\tilde{G}_\infty(H, K)$.

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

$$\psi(X) = \max \left\{ \sum_{j \in X} y(s, j) \mid y \text{ is a feasible } s\text{-}t \text{ flow in } \tilde{G}_\infty(H, K) \right\} \quad (24)$$

holds for each $X \subseteq H$.

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

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

Clearly, the maximum flow value

$$\max \left\{ \sum_{j \in H} y(s, j) \mid y \text{ is a feasible } s\text{-}t \text{ flow in } \tilde{G}_\infty^X \right\}$$

is equal to the right-hand side of (24).

Let ζ be the value of the right-hand side of (21). To prove the lemma it suffices to demonstrate that the value of a maximum flow in \tilde{G}_∞^X is equal to ζ .

We first explain how to construct a feasible s - t flow in \tilde{G}_∞^X with a value equal to ζ . Then we show that the capacity of an s - t cut in \tilde{G}_∞^X is also ζ . Thus, by the maximum-flow minimum-cut theorem (or the weak

duality theorem of LP), the constructed flow is a maximum flow, and it is of the required value ζ .

Split the interval nodes \tilde{W} into two subsets, \tilde{W}_{big} and \tilde{W}_{small} . Set \tilde{W}_{big} consists of intervals I_h that can accommodate all $\eta(X, h)$ jobs that can be processed in I_h . On the other hand, \tilde{W}_{small} consists of all intervals I_h that do not have enough room for processing all $\eta(X, h)$ jobs. Formally,

$$\begin{aligned} \tilde{W}_{\text{big}} &= \{I_h \in \tilde{W} \mid \eta(X, h) \Delta_h \leq \tilde{\mu}^X(I_h, t)\}, \\ \tilde{W}_{\text{small}} &= \{I_h \in \tilde{W} \mid \eta(X, h) \Delta_h > \tilde{\mu}^X(I_h, t)\}. \end{aligned}$$

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

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

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

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

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

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

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

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

Thus, we deduce

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

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

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

and compute its capacity

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

By Lemma 2, the problem (LP) in (23) to be solved in Step 2 can be reduced to the maximum flow problem in network \tilde{G}_b , which is obtained from network $\tilde{G}_\infty(H, K)$ by replacing capacities $\tilde{\mu}(u, v)$ with $\tilde{\mu}_b(u, v)$ given by

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

For a maximum flow x_* in network \tilde{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 an instrumental set Y_* , i.e., a set Y_* satisfying (16). Lemma 3 shows that such a set can be obtained from a minimum s - t cut in \tilde{G}_b . Thus, the problem to be solved in Step 3 is a minimum cut problem in \tilde{G}_b .

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

$$q(S_* \cap H) = \psi(S_* \cap H), \quad q(j) = b(j), \quad j \in H \setminus S_* \quad (25)$$

hold.

Proof. Since x_* is a maximum flow and (S_*, T_*) is a minimum s - t cut, the maximum-flow minimum-cut theorem implies that

$$\sum_{j \in H} x_*(s, j) = \tilde{\mu}_b(S_*, T_*) \quad (26)$$

and $x_*(u, v) = \tilde{\mu}_b(u, v)$ for every $(u, v) \in \tilde{A}(S_*, T_*)$, where

$$\tilde{A}(S_*, T_*) = \{(u, v) \in \tilde{A} \mid u \in S_*, v \in T_*\}.$$

In particular, for each $j \in H \setminus S_* \subseteq T_*$, it holds that $(s, j) \in \tilde{A}(S_*, T_*)$, so that

$$x_*(s, j) = \tilde{\mu}_b(s, j) = b(j), \quad j \in H \setminus S_*, \quad (27)$$

and the second equation in (25) holds.

To verify the first equation in (25), we prove

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

with $X = H \cap S_*$. Let network \tilde{G}_∞^X be as defined in the proof of Lemma 2. The proof of Lemma 2 shows that

$$\psi(X) = \max \left\{ \sum_{j \in H} y(s, j) \mid y \text{ is a feasible } s\text{-}t \text{ flow in } \tilde{G}_\infty^X \right\}. \quad (28)$$

Let $x: \tilde{A} \rightarrow \mathbb{R}_+$ be a feasible s - t flow in \tilde{G}_∞^X such that

$$x(u, v) \leq x_*(u, v), \quad (u, v) \in \tilde{A}, \\ x(s, j) = \begin{cases} x_*(s, j), & \text{if } j \in X; \\ 0, & \text{otherwise.} \end{cases}$$

Such a flow can be easily obtained from x_* ; see, e.g., Ahuja et al. (1993). Notice that $H \setminus S_* = H \setminus (S_* \cap H) = H \setminus X$, so that (27) implies

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

This and (26) yield

$$\sum_{j \in H} x(s, j) = \sum_{j \in H} x_*(s, j) - \sum_{j \in H \setminus X} x_*(s, j) \\ = \tilde{\mu}_b(S_*, T_*) - \sum_{j \in H \setminus X} \tilde{\mu}_b(s, j) = \tilde{\mu}^X(S_*, T_*), \quad (29)$$

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

$$\text{for } (s, j) \in \tilde{A}(S_*, T_*) \text{ with } j \in H \setminus X = H \cap T_*, \\ \tilde{\mu}_b(s, j) = b(j), \quad \tilde{\mu}^X(s, j) = 0, \\ \text{for other arcs } (u, v) \in \tilde{A}(S_*, T_*), \\ \tilde{\mu}_b(u, v) = \tilde{\mu}^X(u, v).$$

It follows from (29) and the maximum-flow minimum-cut theorem (or the weak duality theorem of LP) that x is a maximum flow in \tilde{G}_∞^X , which, together with (28), implies

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

This concludes the proof. \square

In summary, Steps 2 and 3 can be reduced to the maximum flow problem and to the minimum cut problem in network \tilde{G}_b , respectively. Since \tilde{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 \tilde{G}_b can be found in $O(|\tilde{A}|) = O(n^2)$ time; see, e.g., Ahuja et al. (1993), Schrijver (2003). 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 1. The SSP on m parallel machines to minimize the function (1) 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 Φ is rewritten as

$$\hat{\Phi} = \sum_{j=1}^n p(j) f \left(\frac{w(j)}{p(j)} \right), \quad (30)$$

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 nontrivial statement due to Murota (1988) and Nagano and Aihara (2012) that reduces this problem to a quadratic optimization problem.

Theorem 2 (Murota 1988, Nagano and Aihara 2012). *The problem of minimizing the function (30) over a base polyhedron $B(\varphi)$ is equivalent to*

$$\begin{aligned} & \text{minimize} && \sum_{j=1}^n \frac{p(j)^2}{w(j)} \\ & \text{subject to} && \mathbf{p} \in B(\varphi), \end{aligned} \quad (31)$$

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 (31) 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 nonzero costs only on the arcs going out of the source. Exactly such a problem is considered in Gallo et al. (1989) and Hochbaum and Hong (1995), who reduce it to the parametric maximum flow problem and show how to solve it in $O(|V|^3) = O(n^3)$ time. This observation can be summarized as the following statement.

Theorem 3. *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 3 is several orders faster than the best previously known. For a more general problem in Theorem 1, 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 minimum-cost maximum-flow problem with a linear objective cost function in a network similar to \tilde{G}_b . McCormick (1999) shows that the latter problem can be reduced to the parametric maximum flow problem and solved in $O(n^3)$ time.

Remark 4. In Tian et al. (2010) a generalization of the speed scaling problem with *multiple active intervals* is discussed, where each job j is associated with a nonempty set of disjoint time intervals

$$[r(j, 1), d(j, 1)], [r(j, 2), d(j, 2)], \dots, [r(j, s_j), d(j, s_j)].$$

Each job j can be processed only within its active intervals, and it is allowed to use several such intervals for processing job j . Notice that our problem is a special case of the one studied by Tian et al. (2010), with each job having a single time interval: $s_j = 1$ for $j \in N$.

Tian et al. (2010) consider the case with a single machine and job-independent speed cost functions,

and show that the problem can be solved in polynomial time. Next we consider a more general setting, with multiple machines and/or job-dependent speed cost functions, and show that the problem can be also solved in polynomial time by the approach discussed in this section.

The key idea is to use the modified version of the network $G_\infty = (V, A)$ defined as follows. Divide the interval $[\min_{j \in N, 1 \leq k \leq s_j} r(j, k), \max_{j \in N, 1 \leq k \leq s_j} d(j)]$ into subintervals by using $r(j, k)$ and $d(j, k)$ ($j \in N$, $1 \leq k \leq s_j$) as break-points. Denote the resulting subintervals by $I_1, I_2, \dots, I_{\hat{\gamma}}$, $1 \leq \hat{\gamma} \leq 2 \sum_{j \in N} s_j - 1$. The node set of the modified network is given by $V = \{s, t\} \cup N \cup \hat{W}$, where s and t are the source and the sink, N is the set of job nodes, and \hat{W} is the set of interval nodes, i.e., $\hat{W} = \{I_1, I_2, \dots, I_{\hat{\gamma}}\}$. The arc set A is given as $A = A^s \cup \hat{A}^0 \cup A^t$, where

$$\begin{aligned} A^s &= \{(s, j) \mid j \in N\}, \\ \hat{A}^0 &= \{(j, I_h) \mid j \in N, 1 \leq h \leq \hat{\gamma}, \\ &\quad \text{job } j \text{ can be processed in the interval } I_h\}, \\ A^t &= \{(I_h, t) \mid 1 \leq h \leq \hat{\gamma}\}. \end{aligned}$$

Define the arc capacity function $\mu: A \rightarrow \mathbb{R}_+$ in the same way as in Section 2. Note that the number of nodes in the modified network is $O(\sum_{j \in N} s_j)$.

Applying the approach presented in Sections 2–4 (with appropriate adjustments) to the modified network, it can be shown that the following results hold.

Theorem 4. *The speed scaling problem with multiple active intervals on m parallel machines can be solved in $O(n(\sum_{j \in N} s_j)^3)$ time, if speed cost functions are job dependent, and in $O((\sum_{j \in N} s_j)^3)$ time, otherwise.*

5. Solving SSP on a Single Machine

In this section, we explain the implementation details of Algorithm F-Decomp for solving the speed scaling problem in the case of a single machine. As in Section 4, it suffices to explain how to implement Steps 2 and 3 of Algorithm F-Decomp. The approach we adopt in this section differs from the one in Section 4.

In Step 2, we need to solve the linear programming problem (LP) of type (23), where the function $\psi = \psi_K: 2^H \rightarrow \mathbb{R}$ is given by (18). Recall the explicit representation (22) 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

$$[\tau_{\pi(1)-1}, \tau_{\pi(1)}], [\tau_{\pi(2)-1}, \tau_{\pi(2)}], \dots, [\tau_{\pi(v)-1}, \tau_{\pi(v)}],$$

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

$$\left[\min_{j \in H} r(j), \tau_{\pi(1)-1} \right], [\tau_{\pi(1)}, \tau_{\pi(2)-1}], [\tau_{\pi(2)}, \tau_{\pi(3)-1}], \dots, [\tau_{\pi(v-1)}, \tau_{\pi(v)-1}], \left[\tau_{\pi(v)}, \max_{j \in H} d(j) \right].$$

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 v intervals of set W_A , and no job $j \in H$ is scheduled outside the interval $[r(j), d(j)]$. This problem is essentially the same as the one discussed in Hochbaum and Shamir (1990, Section 2) (see also Shih et al. 1991). This is called the problem of minimizing the unweighted number of tardy units, where the objective is to minimize the total (unweighted) compression (or tardy units) $\sum_{j \in H} (b(j) - q(j))$ instead of maximizing $\sum_{j \in H} q(j)$.

For that problem, Hochbaum and Shamir (1990) propose an efficient algorithm based on the UNION-FIND technique and show that the algorithm solves the problem in $O(h + v)$ 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 an instrumental set, i.e., a set $Y_* \subseteq H$ satisfying (16). In scheduling terms, for Problem (LP) the jobs of a set Y_* completely use all intervals available for their processing, while the actual processing times $q(j)$ of the remaining jobs reach their upper bounds $b(j)$. To find an instrumental set Y_* , we make a slight modification of the Hochbaum-Shamir algorithm, without affecting its linear running time. In the description of

the algorithm, the jobs of set H are renumbered by the integers $1, 2, \dots, h$ in nonincreasing order of their release dates, i.e.,

$$r(1) \geq r(2) \geq \dots \geq r(h); \quad (32)$$

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 Hochbaum and Shamir (1990).

(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.

An illustrative example of the behavior of Algorithm HSY is provided in Online Supplement 2. Algorithms similar in nature to Algorithm HSY can be found in Li et al. (2014) and Shioura et al. (2016).

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

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. As a result of 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. \square

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 $t = \max\{d(j) \mid j \in Y^k\}$. 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, $q_*(Y^k) = \psi(Y^k)$. \square

Notice that output Y_* in Step 3 is given as the union of sets Y^1, Y^2, \dots, 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.

For $k \in H$, if $[r(k), d(k)]$ has no idle time in S_k , then k is included in Y^k , and therefore $k \in Y^k \subseteq Y_*$ holds. Hence, if $k \in H \setminus Y_*$, then $[r(k), d(k)]$ has idle time in S_k , implying that $q_*(k) = b(k)$. Thus, set Y_* found by the algorithm satisfies (16), i.e., Y_* is an required instrumental set.

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 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_* , 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 (32) 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 (1) 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 (2005).

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 Yao et al. (1995) and the alternative $O(n^2 \log n)$ approach by Li et al. (2006). Noteworthy, the YDS algorithm is recognized as the most cited result in the speed scaling literature. For a multimachine model, our approach achieves a substantial speed up in comparison with the existing ones by Albers et al. (2015) and Angel et al. (2012).

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 multimachine model with job-dependent speed cost functions f_j . Another enhancement we intend to consider in our future research is the case of nonidentical 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 multimachine models is quite limited, as highlighted in papers by Albers (2009, 2010).

Acknowledgments

The authors thank the referees for their constructive comments and suggestions aimed at improving the paper.

References

- Ahuja RK, Magnanti TL, Orlin JB (1993) *Network Flows: Theory, Algorithms and Applications* (Prentice Hall, Englewood Cliffs, NJ).
- Albers S (2009) Algorithms for energy saving. Albers S, Alt H, Näher S, eds. *Efficient Algorithms*, Lecture Notes in Computer Science, Vol. 5760 (Springer, New York), 173–186.
- Albers S (2010) Energy-efficient algorithms. *Comm. ACM* 53:86–96.
- Albers S, Antoniadis A, Geiner G (2015) On multiprocessor speed scaling with migration. *J. Comput. System Sci.* 81(7):1194–1209.
- Albers S, Müller F, Schmelzer S (2007) Speed scaling on parallel processors. *Algorithmica* 68:404–425.
- Albers S, Bampis E, Letsios D, Lucarelli G, Stotz R (2016) Scheduling on power-heterogeneous processors. *Latin American Sympos. Theoret. Informatics* (Springer, Berlin), 41–54.
- Angel E, Bampis E, Kacem F, Letsios D (2012) Speed scaling on parallel processors with migration. *Euro. Conf. Parallel Processing, Rhodes Island, Greece*, 128–140.
- Bambagini M, Buttazzo G, Bertogna M (2013a) Energy-aware scheduling for tasks with mixed energy requirements. *Proc. 4th Internat. Real-Time Scheduling Open Problems Seminar, Paris, France*.
- Bambagini M, Lelli J, Buttazzo G, Lipari G (2013b) On the energy-aware partitioning of real-time tasks on homogeneous multiprocessor systems. *Proc. 4th Internat. Conf. Energy Aware Comput. Systems Appl., Istanbul, Turkey*, 69–74.

- Bampis E, Letsios D, Lucarelli G (2015) Green scheduling, flows and matchings. *Theor. Comp. Sci.* 579:126–136.
- Bampis E, Kononov V, Letsios D, Lucarelli G, Sviridenko M (2016) Energy-efficient scheduling and routing via randomized rounding. *J. Scheduling*, ePub ahead of print October 28, <https://doi.org/10.1007/s10951-016-0500-2>.
- Fujishige S (1980) Lexicographically optimal base of a polymatroid with respect to a weight vector. *Math. Oper. Res.* 5(2):186–196.
- Fujishige S (2005) *Submodular Functions and Optimization*, Ann. Discr. Math., 2nd ed., Vol. 58 (Elsevier, Amsterdam).
- Gallo G, Grigoriadis MD, Tarjan RE (1989) A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* 18(1):30–55.
- Gordon VS, Tanaev VS (1973) Deadlines in single-stage deterministic scheduling. *Optimization of Systems for Collecting, Transfer and Processing of Analogous and Discrete Data in Local Information Computing Systems* (Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk), 53–58 (in Russian).
- Groenevelt H (1991) Two algorithms for maximizing a separable concave function over a polymatroid feasible region. *Eur. J. Oper. Res.* 54(2):227–236.
- Gupta A, Krishnaswamy R, Pruhs K (2010) Scalably scheduling power-heterogeneous processors. *37th Internat. Colloquium Automata, Languages Program., Bordeaux, France*, 312–323.
- Gupta A, Im S, Krishnaswamy R, Moseley B, Pruhs K (2012) Scheduling heterogeneous processors isn't as easy as you think. *Proc. 23rd Annual ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 1242–1253.
- Hochbaum DS, Hong S-P (1995) About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Math. Program.* 69(1):269–309.
- Hochbaum DS, Shamir R (1990) Minimizing the number of tardy job units under release time constraints. *Discrete Appl. Math.* 28(1):45–57.
- Horn W (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21(1):177–185.
- Ishii H, Martel C, Masuda T, Nishida T (1985) A generalized uniform processor system. *Oper. Res.* 33(2):346–362.
- Karzanov AV (1974) Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl.* 15(2):434–437.
- Li M, Yao AC, Yao FF (2006) Discrete and continuous min-energy schedules for variable voltage processors. *Proc. Nat. Acad. Sci., USA* 103(11):3983–3987.
- Li M, Yao FF, Yuan H (2014) An $O(n^2)$ algorithm for computing optimal continuous voltage schedules. arxiv:1408.5995v1.
- McCormick ST (1999) Fast algorithms for parametric scheduling come from extensions to parametric maximum flow. *Oper. Res.* 47(5):744–756.
- McNaughton R (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6(1):1–12.
- Megiddo N (1974) Optimal flows in networks with multiple sources and sinks. *Math. Program.* 7(1):97–107.
- Murota K (1988) Note on the universal bases of a pair of polymatroids. *J. Oper. Res. Soc. Japan* 31(4):565–573.
- Nagano K, Aihara K (2012) Equivalence of convex minimization problems over base polytopes. *Japan J. Indust. Appl. Math.* 29(3):519–534.
- Nowicki E, Zdrzałka S (1990) A survey of results for sequencing problems with controllable processing times. *Discrete Appl. Math.* 26(2–3):271–287.
- Schrijver A (2003) *Combinatorial Optimization: Polyhedra and Efficiency* (Springer, Berlin).
- Shabtay D, Steiner G (2007) A survey of scheduling with controllable processing times. *Discrete Appl. Math.* 155(13):1643–1666.
- Shakhlevich NV, Strusevich VA (2005) Pre-emptive scheduling problems with controllable processing times. *J. Sched.* 8(3):233–253.
- Shakhlevich NV, Strusevich VA (2008) Preemptive scheduling on uniform parallel machines with controllable job processing times. *Algorithmica* 51(4):451–473.
- Shakhlevich NV, Shioura A, Strusevich VA (2009) Single machine scheduling with controllable processing times by submodular optimization. *Internat. J. Found. Comput. Sci.* 20(2):247–269.
- Shih W-K, Liu JWS, Chung J-Y (1991) Algorithms for scheduling imprecise computations with timing constraints. *SIAM J. Comput.* 20(3):537–552.
- Shioura A, Shakhlevich NV, Strusevich VA (2013) A submodular optimization approach to bicriteria scheduling problems with controllable processing times on parallel machines. *SIAM J. Discrete Math.* 27(1):186–204.
- Shioura A, Shakhlevich NV, Strusevich VA (2015) Decomposition algorithms for submodular optimization with applications to parallel machine scheduling with controllable processing times. *Math. Program.* 153(2):495–534.
- Shioura A, Shakhlevich NV, Strusevich VA (2016) Application of submodular optimization to single machine scheduling with controllable processing times subject to release dates and deadlines. *INFORMS J. Comput.* 28(1):148–161.
- Tian W, Li M, Chena E (2010) Energy optimal schedules for jobs with multiple active intervals. *Theor. Comput. Sci.* 411(3):672–676.
- Venkatachalam V, Franz M (2005) Power reduction techniques for microprocessor systems. *ACM Comput. Surveys* 37(3):195–237.
- Vizing VG, Komzakowa LN, Tarchenko AV (1981) An algorithm for selecting the processing intensity. *Kibernetika* 5:71–74.
- Wu W, Li M, Huang H, Chen E (2012) Speed scaling problems with memory/cache consideration. *Internat. Conf. Theory Appl. Models Comput.*, Lecture Notes in Computer Science, Vol. 7287 (Springer, New York), 412–422.
- Yao FF, Demers AJ, Shenker S (1995) A scheduling model for reduced CPU energy. *Proc. 36th IEEE Sympos. Foundations Comput. Sci., Milwaukee, WI*, 374–382.