Energy Saving Computational Models with Speed Scaling via Submodular Optimization

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 cost (or energy consumption) function. In addition, it handles efficiently the most general models with job-dependent speed cost functions, on a single machine and on multiple parallel machines, which to the best of our knowledge have not been addressed in prior research.

KEYWORDS

Energy minimization, single machine, parallel machines, convex optimization, submodular constraints

1 INTRODUCTION

Scheduling models with variable speeds of machines have been drawing considerable attention since the 1990s. In those models, processors can work at different voltage/frequency levels, achieving 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, and is known under the names "energy-aware scheduling" [7], "energy-saving scheduling" [2], and "green scheduling" [8].

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. [32] provides a fundamental speed scaling algorithmic technique for the most basic model of scheduling n jobs (computational tasks) on 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. A recent implementation of this technique allows reducing its running time to $O\left(n^2\right)$; see [17].

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 results into a $O\left(n^4 \log n\right)$-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 prob-

ISBN: 978-1-941968-15-4

lem (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.

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.). The need to consider individual energy models for tasks dependent either on their computation intensity or on data intensity is widely recognized in the computing community, see, e.g., [7] for an example of a job-dependent energy function.

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.

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 [22, 24]. 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 [14] 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 [25]–[29]. 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 [9]; 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 multi-machine case with the objective function Φ we rely on a non-trivial result in [20, 21] 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 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 (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 $j \in N$. The objective function F becomes

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

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. The famous max-flow min-cut theorem guarantees that if x is a maximum flow and (S, T) is a minimum s-t cut, then

$$\sum_{j \in N} x(s, j) = \mu(S, T) \qquad (4)$$

holds. Moreover, if (4) holds, then x is a maximum flow and (S, T) is a minimum s-t cut. See [1] for details.

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 $1 \le \gamma \le 2n - 1$, be the increasing sequence of distinct numbers in the list $(r(j), d(j) \mid j \in N)$. Introduce the intervals $I_h = [\tau_{h-1}, \tau_h]$, $1 \le h \le \gamma$, and define the set of all intervals $W = \{I_h | 1 \le h \le \gamma\}$. 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 $r(j) \le \tau_h$ and $d(j) \ge \tau_{h+1}$. For a job j, denote the set of the available intervals by $\Gamma(j)$, where

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

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

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

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:

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

As proved in [15], 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. In fact, given a nonnegative vector $\mathbf{p} = (p(1), \ldots, p(n))$, a feasible schedule for processing the jobs of set N on m parallel identical 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 $j \in N$.

For a network with a set of nodes V, an algorithm by Karzanov [16] finds the max-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 interval 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 [18] 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 case of the SSP, the values of $p(j)$ have to be chosen to minimize the function (3). This can be made by solving an appropriate *min-cost/max-flow* problem, i.e., the problem of finding the maximum flow of the smallest cost. Let $x(s, j)$ denote the amount of flow on an arc (s, j), $j \in N$; then the associated cost of that flow is $x(s, j) f_j(w(j)/x(s, j))$, which is a convex function with respect to $x(s, j)$. The cost of flow on all other arcs is set to be zero. Then the SSP reduces to finding a maximum s-t flow x^* in G that minimizes the total cost

$$\sum_{j \in N} x(s, j) f_j\left(\frac{w(j)}{x(s, j)}\right).$$

For our purposes, we need a link of flow problems in networks structurally similar to G to submodular optimization. These issues are discussed in the next section.

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 [10] and [23].

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

$$\varphi(X \cup Y) + \varphi(X \cap Y) \leq \varphi(X) + \varphi(Y),$$

holds for all $X, Y \in 2^N$. Function φ is called *monotone* if $\varphi(X) \leq \varphi(Y)$ holds for every

$X, Y \in 2^N$ with $X \subseteq Y$. For a monotone submodular function $\varphi : 2^N \to \mathbb{R}_+$ such that $\varphi(\emptyset) = 0$, the pair $(2^N, \varphi)$ is called a *polymatroid*, while φ is referred to as the *polymatroid rank function*.

For a vector $\mathbf{p} = (p(1), \ldots, p(n)) \in \mathbb{R}^N$, denote $p(X) = \sum_{j \in X} p(j)$. Given a polymatroid $(2^N, \varphi)$, define two polyhedra

$$\begin{aligned} P_{(+)}(\varphi) = \; & \left\{\mathbf{p} \in \mathbb{R}^N \mid p(X) \le \varphi(X) \right. \\ & \left. X \in 2^N, \; \mathbf{p} \ge 0\right\}; \\ B(\varphi) = \; & \left\{\mathbf{p} \in \mathbb{R}^N \mid \mathbf{p} \in P_{(+)}(\varphi), \right. \\ & \left. p(N) = \varphi(N)\right\} \end{aligned}$$

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 $P_{(+)}(\varphi)$.

Take a network G described in Section 2. Consider the polyhedron

$$\begin{aligned} P = \; & \left\{\mathbf{p} \in \mathbb{R}^N_+ \mid \exists \text{ feasible } s-t \text{ flow } x \right. \\ & \left. \text{in } G \text{ with } p(j) = x(s,j) \text{ for } j \in N\right\}. \end{aligned}$$

It is known (see, e.g., [19, Lemma 4.1], [10, Section 2.2], [13]) that such a polyhedron is a polymatroid polyhedron with a rank function $\varphi : 2^N \to \mathbb{R}_+$ given by

$$\begin{aligned} \varphi(Y) = \; & \max_{Y \subseteq N} \left\{ \sum_{j \in Y} x(s,j) \mid x \text{ is a} \right. \\ & \left. \text{feasible } s-t \text{ flow } x \text{ in } G\right\}. \end{aligned} \tag{7}$$

Furthermore, all possible maximum flows can be characterized as a base polyhedron $B(\varphi)$ of the polymatroid. Thus, in terms of submodular optimization, the SSP on parallel machines can be reformulated as

$$\begin{aligned} \min \; & \sum_{j=1}^{n} p(j) f_j(w(j)/p(j)) \\ \text{s. t.} \; & \mathbf{p} \in B(\varphi). \end{aligned} \tag{8}$$

The problem (8) falls into the category of problems of minimizing a separable convex function with submodular constraints:

$$\begin{aligned} \min \; & \sum_{j=1}^{n} h_j(p(j)) \\ \text{s. t.} \; & \mathbf{p} \in B(\varphi), \end{aligned} \tag{9}$$

where $h_j(\cdot)$ is a convex function and $B(\varphi)$ is a base polyhedron. In particular, for $h_j(p(j)) = p(j) f_j(w(j)/p(j))$, problem (9) coincides with problem (3).

To solve problem (9), we can adapt a decomposition algorithm by Fujishige [9] and Groenevelt [12] (see also Section 8.2 of [10]) given below.

Algorithm Decomp

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

$$\begin{aligned} \min \; & \sum_{j \in N} h_j(p(j)) \\ \text{s. t.} \; & p(N) = \varphi(N), \; \mathbf{p} \ge \mathbf{0}. \end{aligned}$$

Step 2. Find a maximal vector $\mathbf{q} \in \mathbb{R}^N$ satisfying $\mathbf{q} \in P_{(+)}(\varphi)$ and $\mathbf{q} \le \mathbf{b}$.

Step 3. Find a (unique) maximal set $Y_* \subseteq N$ such that $\varphi(Y_*) = q(Y_*)$.

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:

$$\begin{aligned} \min \; & \sum_{j \in Y_*} h_j(p(j)) \\ \text{s. t.} \; & p(X) \le \varphi(X), X \in 2^{Y_*}; \\ & p(Y_*) = \varphi(Y_*); \\ & p(j) \ge 0, \; j \in Y_*. \end{aligned}$$

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

$$\begin{aligned} \min \; & \sum_{j \in N \setminus Y_*} h_j(p(j)) \\ \text{s. t.} \; & p(X) \le \varphi(X \cup Y_*) - \varphi(Y_*), \\ & \qquad X \in 2^{N \setminus Y_*}; \\ & p(N \setminus Y_*) = \varphi(N) - \varphi(Y_*); \\ & p(j) \ge 0, \; j \in N \setminus Y_*. \end{aligned}$$

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

Notice that the problems to be solved in Steps 5 and 6 are of the same structure as the initial problem, and they are solved recursively. The depth of recursion is at most n.

It is easy to verify that vector $\mathbf{b} \in \mathbb{R}^N$ found in Step 1, is such that

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

provided that each function h_j is differentiable. We assume that Step 1 can be implemented in $O(n)$ time. This is, for example, true for the most common case studied in the speed scaling literature when

$$h_j(p(j)) = w(j)^\alpha / p(j)^{\alpha-1} \qquad (10)$$

which corresponds to the power consumption function of the form $f_j(s(j)) = s(j)^\alpha$, where $\alpha > 1$ is a constant. Notice that the case of $\alpha = 3$ models the well-known cube root rule for CMOS devices: the speed is approximately the cube root of the power, or equivalently $f_j(s(j)) = s(j)^3$. For $h_j(p(j))$ of type (10), the solution to Step 1 is given by $p(j) = \varphi(N)w(j)/\sum_{j=1}^{n} w(j)$, $j = 1, \ldots, n$. In scheduling terms this implies that in an optimal solution to the relaxed problem of Step 1, each job j is processed at the same speed $s(j) = w(N)/\varphi(N)$.

Problem (9) and Algorithm Decomp admit the following interpretation in scheduling terms for base polyhedron $B(\varphi)$. The rank function $\varphi(X)$, $X \subseteq N$, specifies the total duration of all time intervals available for processing the jobs of set X. Thus, the values of $b(j), j \in N$, found in Step 1 can be understood as actual processing times of jobs such that their total duration $p(N)$ is equal to the total duration $\varphi(N)$ of all available intervals. This is achieved by processing the jobs at the common speed defined as the total work requirement of all jobs $w(N)$ divided by intervals' length $\varphi(N)$. The values of $b(j)$ are not necessarily feasible durations for all jobs or some subsets of 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 length of all intervals available for processing these jobs. In other words, for each job $j \in Y_*$ its actual duration cannot be further extended.

In the subsequent sections, we explain how to implement the steps of Algorithm Decomp in the case of the speed scaling problems on parallel identical machines and on a single machine. In fact, we only need to focus on the implementation details of Steps 2 and 3. If $T_{23}(k)$ denotes the time that is required for Steps 2 and 3 with k decision variables, $k \leq n$, then, under the assumption regarding the time complexity of Step 1, the overall running time of Algorithm Decomp is $O(nT_{23}(n))$.

In fact, in the scheduling applications discussed in Sections 4 and 5, we describe how to find a set Y_* (see Step 3) which not only satisfies $q(Y_*) = \varphi(Y_*)$, but also satisfies a stronger condition

$$q(Y_*) = \varphi(Y_*); \; q(j) = b(j), \; j \in N \backslash Y_*. \qquad (11)$$

In scheduling terms, this means that for the jobs $j \in Y_*$ the processing times $q(j)$ cannot be further enlarged due to the insufficient processing capacity, while for the jobs $j \in N \backslash Y_*$ the processing times reach their respective current upper bounds $b(j)$.

4 SOLVING SSP ON PARALLEL MACHINES

We start with the problem (8), where the speed cost functions f_j are job-dependent. We show that Steps 2 and 3 of Algorithm Decomp with k decision variables, $k \leq n$, can be implemented in $O(k^3)$ time. For simplicity, we present our analysis for an iteration with n decision variables; see [30] for a complete proof. Suppose that in Step 1 of the algorithm the values $b(j)$, $j \in N$, are found. The problem to be solved in Step 2 can be seen as the max-flow problem in network G_b, obtained from G_∞ by setting the upper bound on the capacity of an arc (s, j) equal to $b(j)$, $j \in N$. This can be done in $O(n^3)$ time, as mentioned in Section 2. Let $y(j)$ be the found flow on arc (s, j), $j \in N$. Then the required value of $q(j)$ is equal to the flow $y(j)$.

ISBN: 978-1-941968-15-4 ©2015 SDIWC

The problem of finding the set Y_* is deeply related to a minimum s-t cut in the network G_b. Indeed, we can find a set $Y_* \subseteq N$ that satisfies (11) from a minimum cut.

Lemma 1 *Let (S,T) be a minimum s-t cut in network G_b. Then, $Y_* = S \cap N$ satisfies (11).*

Proof: Let $x : A \to \mathbb{R}$ be a maximum s-t flow in G_b, so that (4) holds. Modify the capacity function by increasing the capacity $\mu(s,j)$ for arcs (s,j) with $j \in S \cap N$ to $+\infty$ and decreasing the capacity $\mu(s,j)$ for arcs $(s,j) \in E$ with $j \in T \cap N$ to 0. Let the obtained network be called G'. Note that (S,T) is still a minimum s-t cut in G' as it was in G_b, since the arcs (s,j) with $j \in S \cap N$ are not contained in $A(S,T)$ and the arcs (s,j) with $j \in T \cap N$ are contained in $A(S,T)$. On the other hand, the definition of φ in (7) implies that the max-flow value in the modified network G' is equal to $\varphi(S \cap N)$. Hence, we have

$$\mu(S,T) - \sum_{j \in T \cap N} \mu(s,j) = \varphi(S \cap N),$$

where the left-hand side of the equality above is equal to the capacity of the s-t cut (S,T) after the modification of the capacity function is performed. Combining the two previous equalities, we obtain

$$\sum_{j \in N} x(s,j) = \varphi(S \cap N) + \sum_{j \in T \cap N} \mu(s,j),$$

from which it follows that

$$\sum_{j \in S \cap N} x(s,j) = \varphi(S \cap N);$$
$$x(s,j) = \mu(s,j),\ j \in T \cap N,$$

since x is a feasible s-t flow and therefore satisfies

$$\sum_{j \in X} x(s,j) \le \varphi(X),\ X \subseteq N;$$
$$x(s,j) \le \mu(s,j),\ j \in N.$$

This implies that set $Y_* = S \cap N$ satisfies (11).1qed

Given a maximum s-t flow in network G, a minimum s-t cut can be found in $O(|A|) = O(n^2)$ time; see, e.g., [1] and [23].
Thus, each of n iterations of Algorithm Decomp requires $O(n^3)$ time. Applying this algorithm, we find the actual processing times $p(j)$ of the jobs, and the optimal speeds can be defined 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 is $\hat{\Phi}$ given by

$$\hat{\Phi} = \sum_{j=1}^{n} p(j) f\left(\frac{w(j)}{p(j)}\right) \qquad (12)$$

We show that in this case, the problem can be solved faster. The basis of our reasoning is a non-trivial statement due to [20] and [21] that reduces the problem of minimizing (12) to the problem of quadratic optimization.

Theorem 2 ***Suppose that function f in the definition (12) of $\hat{\Phi}$ is differentiable and strictly convex. Then, the problem of minimizing the function $\hat{\Phi}(\mathbf{p})$ under the constraint $\mathbf{p} \in B(\varphi)$ is equivalent to the problem of minimizing a separable quadratic convex function $\sum_{j=1}^{n} p(j)^2/w(j)$ under the same constraint.***

Thus, to minimize function $\hat{\Phi}$ we do not need Algorithm Decomp. Instead we can solve the problem of minimizing a separable quadratic convex 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 convex cost function, with nonzero costs only on the arcs linked to the source. Exactly such a problem is considered in [11] and [13], where the problem is

reduced to the parametric max-flow problem in network G and can be solved in $O(n^3)$ time.

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 of $O(n^3)$ established in Theorem 3 is considerably faster than the best previously known running time mentioned in Section 1. For a more general problem in Theorem 1, with job-dependent speed costs, we are not aware of any prior results.

5 SOLVING SSP ON SINGLE MACHINE

In this section, we discuss the implementation of Steps 2 and 3 in an iteration of Algorithm Decomp in the case of a single machine. As in Section 4, we present our analysis for an iteration with n decision variables. In scheduling terms, the problem to be solved in Steps 2 and 3 is a version of a single machine problem with controllable processing times to minimize total compression time. More specifically, the problem to be solved in Step 2 is that of determining the actual processing times $q(j)$ of jobs of set N to maximize the total processing time $\sum_{j\in N} q(j)$, provided that no job j is scheduled outside the interval $[r(j), d(j)]$. Clearly, to maximize $\sum_{j\in N} q(j)$ is the same as to minimize the total compression time $\sum_{j\in N} z(j)$, where $z(j) = w(j) - q(j)$.

The latter problem with controllable processing times can be solved by algorithms developed by [14] and [31]. In particular, the algorithm by [14] uses the UNION-FIND technique and guarantees that the actual processing times of all jobs and the corresponding optimal schedule are found in $O(n)$ time, provided that the jobs are numbered in non-increasing order of their release dates, i.e.,

$$r(1) \geq r(2) \geq \cdots \geq r(n); \qquad (13)$$

additionally, we assume that if $r(j) = r(j+1)$ for some $j \in N$ then $d(j) \leq d(j+1)$ holds.

The Hochbaum-Shamir 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 $j \in N$ is placed into the current partial schedule to fill the available time intervals consecutively, from right to left, starting from the rightmost 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.

For our purposes, however, we not only need the optimal values $q(j)$ of the processing times, but also a set $Y_* \subseteq N$ with $q(Y_*) = \varphi(Y_*)$. This can be achieved by a slight modification of the Hochbaum-Shamir algorithm, which does not affect its linear running time. In addition, for the resulting set Y_* condition (11) holds.

Recall that the meaning of the rank function φ is such that $\varphi(X)$ is the total duration of all intervals available for processing the jobs of set X. A job that belongs to the set Y_* is called *critical*. The length of a critical job cannot be extended (even ignoring its upper bound) without compromising feasibility of the schedule. If a job j is not critical, 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 $b(j)$). Based on this idea, we can obtain a maximal Y_* satisfying the condition (11) in $O(n)$ time as well. See [29] where a similar adaptation of the Hochbaum-Shamir algorithm is presented to be used to as a subroutine for solving a certain single machine scheduling problem with controllable processing times.

We explain the implementation details of how to compute vector q and set Y_* in Steps 2 and 3 of Algorithm Decomp in the case of a single machine.

In the description of the algorithm below, the jobs are assumed to be numbered in ac-

cordance with (13). For a feasible schedule, an interval during which the machine is permanently busy 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 $Y_*^0 := \varnothing$.

Step 2. For each job v **from** 1 **to** n **do**

(a) Schedule job v **in accordance with the algorithm by [14].**

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

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

Each lemma below is applied to schedule S_v**, which is the schedule found in Step 2(a) of Algorithm HSY for the jobs** $1, \ldots, v$**.**

Lemma 2 ***In schedule*** S_v ***any job*** $j \leq v$ ***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_v **such that job** j, $j \leq v$, **is processed in each of these blocks. Due to the feasibility of schedule** S_v, **we have that** $r(j) < t_2 < t_3 < d(j)$**, 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 3 ***If the interval*** $[r(v), d(v)]$ ***has no idle time in schedule*** S_v***, then*** Y^v ***satisfies*** $q(Y^v) = \varphi(Y^v)$**.**

Proof: Lemma 2 implies that in Step 2(b) of Algorithm HSY, Y^v **is the set of jobs that start and complete in block** B^v**. Since job** v **has the smallest release date among all jobs in schedule** S_v **and the interval** $[r(v), d(v)]$ **has no idle time, it follows that block** B^v **contains the interval** $\hat{I} = [r(v), t]$**, where** $t = \max\{d(j) | j \in Y^v\}$**. Let** δ **denote the total length of all busy subintervals within the interval** $\hat{I}$**. Then** $q(Y^v) = t - r(v) - \delta$**. On the other hand, no job of set** Y^v **can start before time** $r(v)$**, complete after time** t **and be assigned to the intervals which are already busy, so that** $\varphi(Y^v) = t - r(v) - \delta$**. Thus,** $q(Y^v) = \varphi(Y^v)$**.** ■

Note that the set Y_* **which is output in Step 3 of Algorithm HSY is given as the union of sets** $Y^1, Y^2, \ldots, Y^n$**, and each** Y^v $(v = 1, 2, \ldots, n)$ **satisfies** $q(Y^v) = \varphi(Y^v)$ **by Lemma 3. By the submodularity of** φ**, the equality** $q(Y_*) = \varphi(Y_*)$ **holds.**
For job $v \in H$**, if** $[r(v), d(v)]$ **has no idle time in** S_v**, then** v **is included in** Y^v**, and therefore** $v \in Y^v \subseteq Y_*$ **holds. Hence, if** $v \in H \setminus Y_*$**, then** $[r(v), d(v)]$ **has idle time in** S_v**, implying that** $q(v) = b(v)$**. Thus, set** Y_* **found by the algorithm satisfies the condition (11).**
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 v **in the original intervals of availability, the required block** B^v **(the set of intervals that contains the latest interval for processing job** v**) will be found (see Step 2(b)). To be able to determine the set** Y^v **of the jobs in block** B^v**, 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^v **are added to set** Y_*^v**, 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.
Algorithm HSY can be used to implement Steps 2 and 3 of Algorithm Decomp. The jobs can be renumbered in accordance with (13) once in $O(n \log n)$ time, so that the overall running time of Algorithm Decomp is $O(n \log n + nT_{23}(n)) = O(n^2)$, and the following statement holds.

Theorem 4 *The SSP on a single machine to minimize the function (1) can be solved in* $O(n^2)$ *time.*

6 CONCLUSION

In our study, we have provided a new methodology for solving the speed scaling problem (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 [10].
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, matching the best known running time given in [17] for the special case of the problem with a speed cost function f common for all jobs. In fact, all previously known algorithms for single machine speed scaling, including the famous YDS algorithm can be seen as implementations of Algorithm Decomp; see [30] for details.
For a multi-machine model, our approach achieves a substantial speed up in comparison with the existing ones by [4] and [6].
The proposed methodology provides a new insight into the underlying optimization model and demonstrates a potential for handling advanced features of enhanced models. It delivers the first efficient solution algorithm to the most general multi-machine model with job-dependent speed cost functions f_j.

REFERENCES

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

[2] S. Albers, "Algorithms for energy saving," Lect. Notes Comp. Sci., vol. 5760, pp. 173–186, 2009.

[3] S. Albers, "Energy-efficient algorithms," Comm. ACM, vol. 53, pp. 86–96, 2010.

[4] S. Albers, A. Antoniadis, and G. Geiner, "On multiprocessor speed scaling with migration," in Proceedings of SPAA, 2012, pp. 279–288.

[5] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," Algorithmica, vol. 68, pp. 404–425, 2012.

[6] E. Angel, E. Bampis, F. Kacem, and D. Letsios, "Speed scaling on parallel processors with migration". Lect. Notes Comp. Sci., vol. 7484, pp. 128–140, 2012.

[7] M. Bambagini, J. Lelli, G. Buttazzo, and G. Lipari, "On the energy-aware partitioning of real-time tasks on homogeneous multi-processor systems," in Proc. 4th Int. Conf. Energy Aware Comput., pp. 69–74, 2014.

[8] E. Bampis, D. Letsios, and G. Lucarelli, "Green scheduling, flows and matchings," Theor. Comp. Sci., vol. 579, pp. 126–136, 2015.

[9] S. Fujishige, "Lexicographically optimal base of a polymatroid with respect to a weight vector," Math. Oper. Res. vol. 5, pp. 186–196, 1980.

[10] S. Fujishige, Submodular Functions and Optimization, 2nd ed., Ann. Discr. Math. vol. 58, Amsterdam: Elsevier, 2005.

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

[12] H. Groenevelt, "Two algorithms for maximizing a separable concave function over a polymatroid feasible region," Eur. J. Oper. Res., vol. 54, pp. 227–236, 1991.

[13] D.S. Hochbaum and S.-P. Hong, "About strongly polynomial time algorithms for

quadratic optimization over submodular constraints," Math. Progr., vol. 69, pp. 269–309, 1995.

[14] D.S. Hochbaum and R. Shamir, "Minimizing the number of tardy job unit under release time constraints," Discr. Appl. Math., vol. 28, pp. 45–57, 1990.

[15] W. Horn, "Some simple scheduling algorithms," Naval Res. Logist. Quart., vol. 21, pp. 177–185, 1974.

[16] A.V. Karzanov, "Determining the maximal flow in an network by the method of preflows," Soviet Math. Doklady, vol. 15, pp. 434–437, 1974.

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

[18] R. McNaughton, "Scheduling with deadlines and loss functions," Manag. Sci., vol. 12, pp. 1–12, 1959.

[19] N. Megiddo, "Optimal flows in networks with multiple sources and sinks," Math. Progr., vol. 7, pp. 97–107, 1974.

[20] K. Murota, "Note on the universal bases of a pair of polymatroids," J. Oper. Res. Soc. Japan, vol. 31, pp. 565–573, 1988.

[21] K. Nagano and K. Aihara, "Equivalence of convex minimization problems over base polytopes," Japan J. Indust. Appl. Math., vol. 29, pp. 519–534, 2012.

[22] E. Nowicki and S. Zdrzałka, "A survey of results for sequencing problems with controllable processing times," Discr. Appl. Math., vol. 26, pp. 271–287, 1990.

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

[24] D. Shabtay and G. Steiner, "A survey of scheduling with controllable processing times," Discr. Appl. Math.,vol. 155, pp. 1643–1666, 2007.

[25] N.V. Shakhlevich and V.A. Strusevich, "Preemptive scheduling problems with controllable processing times," J. Sched. vol. 8, pp. 233–253, 2005.

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

[27] 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. Discr. Math., vol. 27, pp. 186–204, 2013.

[28] 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. Progr., vol. 153, pp. 495–534, 2015.

[29] A. Shioura, N.V. Shakhlevich, and V.A. Strusevich, "Application of submodular optimization to single machine scheduling with controllable processing times subject to release dates and deadlines" INFORMS J. Comput., in press, 2015.

[30] A. Shioura, N.V. Shakhlevich, and V.A. Strusevich. "Machine speed scaling by adapting methods for convex optimization with submodular constraints". Report SORG-03-2015

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

[32] F.F. Yao, A.J. Demers, and S. Shenker," A scheduling model for reduced CPU energy," Proc. 36th IEEE Symp. Found. Comput. Sci. pp. 374–382, 1995.