Green Scheduling, Flows and Matchings

Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

Abstract

Recently, optimal combinatorial algorithms have been presented for the energy minimization *multiprocessor speed-scaling problem with migrations* [Albers et al., SPAA 2011], [Angel et al., Euro-Par 2012]. These algorithms use repeated maximum-flow computations that allow the partition of the set of jobs into subsets in which all the jobs are executed at the same speed. The optimality of these algorithms is based on a series of technical lemmas showing that this partition and the corresponding speeds lead to the minimization of the energy consumption. In this paper, we show that both the algorithms and their analysis can be greatly simplified. In order to do this, we formulate the problem as a convex cost flow problem in an appropriate flow network. Furthermore, we show that our approach is useful to solve other problems in the dynamic speed-scaling setting. As an example, we consider the *preemptive open-shop speed-scaling problem* and we propose a polynomial-time algorithm for finding an optimal solution based on the computation of convex cost flows. We also propose a polynomial-time algorithm for minimizing a linear combination of the sum of the completion times of the jobs and the total energy consumption, for the *non-preemptive multiprocessor speed-scaling problem.* Instead of using convex cost flows, our algorithm is based on the computation of a minimum weighted maximum matching in an appropriate bipartite graph.

Keywords: speed-scaling, polynomial-time algorithms, convex cost flows, weighted matchings

1. Introduction

In the last few years, a series of papers deal with the minimization of the energy consumption in the area of scheduling (see the recent surveys [3] and [4] and the references therein). One of the most studied models in this context is the speed-scaling model in which a set of tasks has to be executed on one

Email addresses: Evripidis.Bampis@lip6.fr (Evripidis Bampis), Dimitrios.Letsios@lip6.fr (Dimitrios Letsios), Giorgio.Lucarelli@lip6.fr (Giorgio Lucarelli)

A preliminary version of this work has been published in the Proceedings of 23rd International Symposium on Algorithms and Computation (ISAAC 2012).

or more processors whose speed may change dynamically during the schedule. Hence, the scheduler has to decide not only the job to execute at any given time, but also the speed of the processor(s) in order to satisfy some level of Quality of Service (QoS), while at the same time to minimize the overall energy consumption. In speed-scaling, power is usually defined as a convex function of the speed and the energy is power integrated over time. Intuitively, the higher is the speed, the better is the performance in terms of QoS, but the higher is the consumption of energy.

The first theoretical result in this area has been proposed in the seminal paper of Yao et al. [21], where the authors considered the energy minimization problem when a set of jobs, each one specified by its processing volume (work), its release date and its strict deadline, has to be scheduled on a single speed-scalable processor. They proposed an algorithm that solves the problem optimally in polynomial time, when the preemption of the jobs, i.e. the possibility to interrupt the execution of a job and resume it later, is allowed. Since then, different problems have been studied taking into account the energy consumption, mainly in the single processor case (e.g., [6, 20]), but more recently in the multiprocessor case as well (e.g., [5, 7, 10, 17]). Different algorithmic techniques have been used in order to optimally solve different speed-scaling scheduling problems, including the use of greedy algorithms, dynamic programming, convex programming, and more recently, maximum flows.

In this paper, we show that the use of convex cost flow computations may lead to polynomial-time algorithms for basic speed-scaling scheduling problems. This adds a new tool for solving speed-scaling scheduling problems. More precisely, we first revisit the *multiprocessor speed-scaling problem with migrations* studied in [5, 7, 10], and we show that it can be solved easily using a convex cost flow formulation, simplifying both the existing algorithms and their proofs of optimality. This problem is the same as the one considered in [21], except that now there are m processors on which the jobs have to be executed and that the execution of an interrupted job may be continued on the same or on another processor (i.e., the migration of jobs is allowed). In [10], Bingham and Greenstreet presented an optimal algorithm for this problem based on the use of the Ellipsoid method. In [5], Albers et al. proposed an $O(n^2 f(n))$-time combinatorial algorithm, where $f(n)$ is the time to compute a maximum flow in a graph. This algorithm is based on a series of maximum flow computations each one of them determining a set of jobs that will be executed with the same speed. Independently, in [7] a similar approach has been presented. Our method simplifies the proof of optimality of the algorithm proposed in [5, 7] in the price of the use as black box of a more sophisticated flow algorithm, namely a computation of a convex cost flow. This flow provides the speeds for all jobs in an optimal solution, i.e., the speeds that minimize the energy consumption which corresponds to the convex cost of the flow.

We also consider the *preemptive open-shop speed-scaling problem.* In this problem, there are m speed-scalable processors and n jobs, and every job is composed by a set of operations, at most one per processor, and every operation is characterized by its processing volume. There is no order in the execution of the operations but two operations of the same job cannot be executed in parallel. The jobs are all available at the same time and there is a common deadline. This is the first attempt to study a speed-scaling problem in a shop scheduling environment. Our algorithm is based again on a transformation to a convex cost flow problem. However, we are not able to calculate in advance the total flow in an optimal solution for our problem, but we propose a generic procedure that uses several convex cost flow computations. In fact, this procedure is a kind of binary search that determines at the same time the total flow and the energy consumption in an optimal solution. Note that this procedure can be used directly for other problems in the speed-scaling setting which can be formulated as a convex cost flow problem.

Finally, we propose a polynomial-time algorithm for minimizing a linear combination of the sum of the completion times of the jobs and the total energy consumption, for the *non-preemptive multiprocessor speed-scaling problem*. Here, we are given m processors and n jobs, each one characterized by its processing volume, while the preemption of the jobs is not allowed. The proposed algorithm is based on the computation of a minimum weighted maximum matching in an appropriate bipartite graph. Notice that in [20] the complexity of the single-processor speed-scaling problem with preemptions where the jobs are subject to release dates has been left open. Our result makes progress towards answering this challenging question.

In Section 2 we present the notation concerning the speed-scaling mechanism as well as the convex cost flow and the minimum weighted maximum matching problems. In Sections 3, 4 and 5 we deal with the three speed-scaling scheduling problems mentioned above. In each section, we formally define the studied problem and we give the related work and our approach for it. In Section 6 we discuss more problems that can be solved using the idea of convex cost flows and we propose some interesting open questions.

2. Preliminaries

In most works in the speed-scaling area (see for example [21]), if a processor operates at a speed $s(t)$ at time t, then its energy consumption rate (power) is equal to $P(s(t)) = s(t)^\alpha$, where $\alpha > 1$ is a constant usually between two and three. In this paper we consider the more general model (see for example [9]) in which the power $P(s(t))$ of a processor is any differentiable convex function of the speed $s(t)$. Then the energy consumption is equal to the power integrated over time. Using a standard exchange argument and based on the convexity of the speed-to-power function, it can be shown that each job/operation runs at a constant speed in any optimal schedule for the considered scheduling problems (see for example [21]).

An instance of the convex cost flow problem consists of a network $N = (V, A)$, where V is a set of nodes and $A \subseteq V \times V$ is a set of arcs between the nodes. Each arc $(u, v) \in A$ is associated with a capacity $c_{u,v} \geq 0$ and a cost function $\kappa_{u,v}(f) \geq 0$, where $f \geq 0$. The function $\kappa_{u,v}(f)$ is convex with respect to f and it represents the cost incurred if f units of flow pass through the arc

(u, v). Moreover, we are given an amount of flow $\mathcal{F}$, a source node $s \in V$ and a destination node $t \in V$. The objective is to route the amount of flow $\mathcal{F}$ from s to t so that the total cost is minimized and the amount of flow that crosses each edge (u, v) does not exceed the capacity $c_{u,v}$, for each $(u, v) \in A$. The convex cost flow problem can be efficiently solved in $O(|A| \log(\max\{\mathcal{F}, c_{max}\})(|A| + |V| \log |V|))$ time, where $c_{max} = \max_{(u,v)\in A}\{c_{u,v}\}$ (see for example [2]).

An instance of a minimum weighted maximum bipartite matching problem consists of a bipartite graph $G = (V, U; A)$, where each edge $e \in A$ has a weight $\kappa_e \geq 0$. A matching M in G is a subset of edges, i.e. $M \subseteq A$, such that no two edges in M have a common endpoint, while the weight of the matching M is equal to $\sum_{e\in M} \kappa_e$. A matching of maximum cardinality is a matching that contains the maximum number of edges among all possible matchings in G. The objective is to find the matching with the minimum weight among the matchings of maximum cardinality. There exists an algorithm for finding such a matching in $O(|V|(|A| + |V| \log |V|))$ time (see for example [2]).

3. Energy Minimization on Multiprocessors with Migrations

The problem. We consider the scheduling problem of minimizing the energy consumption of a set of n jobs that have to be executed on m parallel processors, where each job J_j is characterized by a processing volume (or work) w_j, a release date r_j and a deadline d_j. In this setting, the preemption and the migration of the jobs are allowed, i.e., a job may suspend its execution and continue on the same or another processor, later from the point of suspension. A convex speed-to-power function $P(s)$ defines the energy consumption rate of any processor running at speed $s \geq 0$. By extending the Graham's classical three-field notation [16] in the speed-scaling setting, we denote this problem as $S|pmtn, r_j, d_j|E$.

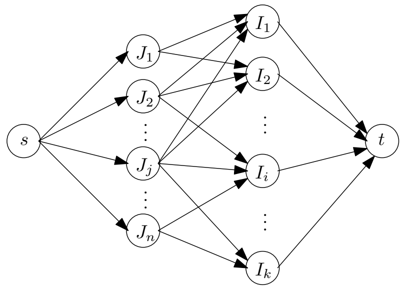
Previous results. This problem is an extension in the speed-scaling setting of one basic problem in scheduling theory, the well-known $P|pmtn, r_j, d_j|-$ problem. In this problem, we are given a set of n jobs that have to be executed on a set of m parallel identical processors, while preemption and migration of jobs are permitted. Each job J_j has a processing time p_j, a release date r_j and a deadline d_j. The objective is to either construct a feasible schedule in which every job J_j is executed during its interval $[r_j, d_j]$, or decide that such a schedule does not exist. The $P|pmtn, r_j, d_j|-$ problem can be solved in polynomial time (see [11]).

Polynomial-time algorithms for finding an optimal solution for $S|pmtn, r_j, d_j|E$, known as the *multiprocessor speed-scaling problem with migration*, have been proposed by Bingham and Greenstreet [10], Albers et al. [5] and Angel et al. [7]. The algorithm in [10] is based on the use of the Ellipsoid method. As the complexity of the Ellipsoid algorithm is high for practical applications, [5] and [7] proposed purely combinatorial algorithms. These algorithms use repeated computations of maximum flows in appropriate flow networks, in order to determine a partition of the set of jobs into subsets in which all the jobs are executed with the same speed. When the speed of such a subset of jobs is determined, these jobs as well as the corresponding time-intervals and processors are removed from the flow network and the process continuous until no job remains unscheduled. At the end, every job is associated with a unique speed and thus an execution time. The final schedule can be produced by applying the algorithm of McNaughton [19]. The optimality of the algorithms in [5] and [7] is based on a series of technical lemmas showing that this partition and the corresponding speeds lead to the minimization of the energy consumption.

3.1. Our Approach

The rough idea of our algorithm is the following: we first formulate the problem $S|pmtn, r_j, d_j|E$ as a convex cost flow problem. An optimal convex cost flow allows us to get the optimal speed s_j for every job J_j, and thus its total execution time $t_j = \frac{w_j}{s_j}$. Then, given the execution times of the jobs, the algorithm constructs a feasible schedule by applying a polynomial-time algorithm for $P|pmtn, r_j, d_j|-$.

Convex cost flow formulation. We consider that the time is partitioned into intervals defined by the release dates and the deadlines of jobs. That is, we define the time points $t_0, t_1, \ldots, t_k$, in increasing order, where each t_i corresponds to either a release date or a deadline, so that for each release date and deadline of job there is a corresponding t_i. Then, we define the intervals $I_i = [t_{i-1}, t_i]$, for $1 \leq i \leq k$, and we denote by $|I_i|$ the length of I_i. We call a job J_j alive in a given interval I_i, if $I_i \subseteq [r_j, d_j]$. The number of alive jobs in interval I_i is denoted by $A(I_i)$.



	N_s	
arc	c	κ
(s, J_j)	$+\infty$	$f_{s,J_j} \cdot P(\frac{w_j}{f_{s,J_j}})$
(J_j, I_i)	$\lvert I_i \rvert$	0
(I_i, t)	$m\lvert I_i \rvert$	0

Figure 1: The flow network N_s for $S|pmtn, r_j, d_j|E$.

Then, in the flow network N_s for $S|pmtn, r_j, d_j|E$ (see Figure 1), we introduce a source node s, a destination node t, a node for each job J_j, $1 \leq j \leq n$, and a node for each interval I_i, $1 \leq i \leq k$. For each j, $1 \leq j \leq n$, we add an arc (s, J_j) and, for each i, $1 \leq i \leq k$, we add an arc (I_i, t). If the job J_j, $1 \leq j \leq n$, is alive during the interval I_i, $1 \leq i \leq k$, we introduce an arc from the node J_j

to the node I_i. The capacity of the arc (u,v) is

$$c_{u,v} = \begin{cases} +\infty & \text{if } u = s \text{ and } v = J_j \\ |I_i| & \text{if } u = J_j \text{ and } v = I_i \\ m|I_i| & \text{if } u = I_i \text{ and } v = t \end{cases}$$

If an amount of flow $f_{u,v}$ passes through the arc (u,v) of N_s, then the cost function of the arc is defined as

$$\kappa_{u,v}(f_{u,v}) = \begin{cases} f_{u,v} \cdot P(\frac{w_j}{f_{u,v}}) & \text{if } u = s \text{ and } v = J_j \\ 0 & \text{otherwise} \end{cases}$$

In the network N_s, if an amount $f_{u,v}$ of flow passes through the arc $(u,v) = (s, J_j)$, then $f_{u,v}$ corresponds to the execution time of job J_j, $\frac{w_j}{f_{u,v}}$ corresponds to the speed of J_j and $f_{u,v} \cdot P(\frac{w_j}{f_{u,v}})$ is the energy consumed for the execution of J_j. Furthermore, the flow passing through an edge (J_j, I_i) (resp. an edge (I_i, t)) represents the execution time of the job J_j (resp. the execution time of all jobs) during the interval I_i. Hence, the total flow that leaves the source node corresponds to the total execution time of all jobs. In [5], it was shown that the total execution time of all jobs in an optimal schedule for $S|pmtn, r_j, d_j|E$ can be easily computed using the following lemma, whose proof is based on the convexity of the speed-to-power function.

Lemma 1. [5] *In an optimal schedule for $S|pmtn, r_j, d_j|E$, where each job J_j, $1 \le j \le n$, is executed with speed s_j, the total execution time, $\mathcal{T}$, of all jobs is*

$$\mathcal{T} = \sum_{j=1}^{n} \frac{w_j}{s_j} = \sum_{i=1}^{k} \Big(\min\{m, A(I_i)\} \cdot |I_i| \Big)$$

The above lemma gives the total amount of flow that has to be sent from the source node to the destination node, concluding the formulation of $S|pmtn, r_j, d_j|E$ as a convex cost flow problem.

The algorithm and its optimality. Our algorithm for $S|pmtn, r_j, d_j|E$ can be summarized as follows.

Algorithm 1

```
1: Construct the flow network $N_s$;
2: Find a convex cost flow $\mathcal{F}$ of value $\sum_{i=1}^{k}(\min\{m, A(I_i)\} \cdot |I_i|)$ in $N_s$;
3: Determine the execution time of each job;
4: Apply a polynomial-time algorithm for $P|pmtn, r_j, d_j|-$ to find a feasible
   schedule;
```

In order to establish the optimality of our algorithm, we need the following lemma whose proof can be found in [19]. The lemma concerns the problem $P|pmtn, r_j = 0, d_j = d|-$ in which all jobs have a common release date and a common deadline.

Lemma 2. [19] *An instance of $P|pmtn, r_j = 0, d_j = d|-$ is feasible if and only if (i) $p_j \leq d$, for each $1 \leq j \leq n$, and (ii) $\sum_{j=1}^{n} p_j \leq m \cdot d$.*

Theorem 1. *Algorithm 1 finds an optimal schedule for $S|pmtn, r_j, d_j|E$ in $O(n^4 \log(m \cdot L))$ time, where $L = \max_{1 \leq j \leq n}\{d_j\}$.*

Proof. We first prove that there is a feasible schedule for $S|pmtn, r_j, d_j|E$ with total execution time of all jobs equal to $\mathcal{T} = \sum_{i=1}^{k}(\min\{m, A(I_i)\} \cdot |I_i|)$, if and only if, there is a feasible flow of value $\mathcal{T}$ in N_s.

Assume that there exists a feasible schedule for $S|pmtn, r_j, d_j|E$ with total execution time equal to $\mathcal{T}$. Let e_j be the execution time of job J_j in this schedule and let $e_j(I_i)$ be the total time that J_j is being processed by some processor during the interval I_i. Consider the flow $\mathcal{F}$ in N_s which is defined as follows:

$$f_{u,v} = \begin{cases} e_j & \text{if } u = s \text{ and } v = J_j \\ e_j(I_i) & \text{if } u = J_j \text{ and } v = I_i \\ \sum_{J_j : I_i \subseteq [r_j, d_j]} e_j(I_i) & \text{if } u = I_i \text{ and } v = t \end{cases}$$

Since the parallel execution of a job is not permitted, for each job J_j and each interval $I_i \subseteq [r_j, d_j]$ it holds that $e_j(I_i) \leq |I_i|$. Moreover, it holds that (i) each processor can operate for at most $|I_i|$ units of time during an interval I_i, $1 \leq i \leq k$, (ii) each processor can execute at most one job at each time and (iii) there are m available processors. Therefore, for each $1 \leq i \leq k$, it holds that $\sum_{J_j : I_i \subseteq [r_j, d_j]} e_j(I_i) \leq \min\{m, A(I_i)\} \cdot |I_i|$. Hence, $\mathcal{F}$ is a feasible flow in N_s because it does not exceed the capacity of any arc.

Assume now that there is a feasible flow $\mathcal{F}$ of value $\mathcal{T}$ in N_s. In order to define a feasible schedule $\mathcal{S}$ for $S|pmtn, r_j, d_j|E$, we assign to each job J_j, $1 \leq j \leq n$, a speed $s_j = \frac{w_j}{f_{s,J_j}}$. So, the total execution time of J_j is f_{s,J_j}. Moreover, for each job J_j, $1 \leq j \leq n$, and each interval $I_i \subseteq [r_j, d_j]$, we set the execution time of J_j during I_i to be f_{J_j,I_i}. Consider now any interval I_i, $1 \leq i \leq k$, in $\mathcal{S}$ and let $e_j(I_i)$ be the total time that J_j is processed by any processor during I_i. Since $\mathcal{F}$ is a feasible flow, it holds that $e_j(I_i) = f_{J_j,I_i} \leq |I_i|$ and $\sum_{J_j : I_i \subseteq [r_j, d_j]} e_j(I_i) = \sum_{i=1}^{k} f_{I_i,t} \leq \min\{m, A(I_i)\} \cdot |I_i|$. By Lemma 2, for each interval I_i, $1 \leq i \leq k$, there is a feasible schedule during I_i, and hence there is a feasible schedule $\mathcal{S}$ for the whole instance of $S|pmtn, r_j, d_j|E$.

Next, we establish the optimality of our algorithm. By the convex cost flow computation, the algorithm finds a flow of value $\sum_{i=1}^{k}(\min\{m, A(I_i)\} \cdot |I_i|)$ such that the term $\sum_{j=1}^{n} f_{s,J_j} \cdot P(\frac{w_j}{f_{s,J_j}})$ is minimized. By our previous claim, we may define a feasible schedule of total execution time $\sum_{i=1}^{k}(\min\{m, A(I_i)\} \cdot |I_i|)$ where each each job J_j is assigned a speed $\frac{w_j}{f_{s,J_j}}$, $1 \leq j \leq n$. Since the energy consumption of this schedule is equal to $\sum_{j=1}^{n} f_{s,J_j} \cdot P(\frac{w_j}{f_{s,J_j}})$, it is a minimum energy schedule among the schedules of total execution time $\sum_{j=1}^{n} f_{s,J_j} \cdot P(\frac{w_j}{f_{s,J_j}})$. By Lemma 1, an optimal schedule for $S|pmtn, r_j, d_j|E$ has total execution time $\sum_{i=1}^{k}(\min\{m, A(I_i)\} \cdot |I_i|)$. Hence, the schedule returned by the algorithm is optimal for the $S|pmtn, r_j, d_j|E$ problem.

The complexity of our algorithm is dominated by Line 2 where a convex cost flow is sought. The flow network N_s constructed by the algorithm contains $O(n)$ nodes and $O(n^2)$ arcs. Moreover, the maximum capacity among all arcs is at most $m \cdot L$, where $L = \max_{1 \leq j \leq n}\{d_j\}$, while by Lemma 1 the total amount of the flow that crosses the network is at most $m \cdot L$. Hence, the overall complexity of our algorithm is $O(n^2 \log(m \cdot L)(n^2 + n \log n)) = O(n^4 \log(m \cdot L))$. Note that Line 4 runs in $O(n^2)$ time [11]. $\square$

4. Energy Minimization on an Open-Shop with Preemptions

In this section we study the well-known preemptive open-shop problem in the speed-scaling setting. We follow again the idea of transforming the problem as a convex cost flow problem. For this problem we are not able to specify in advance the value of flow which again corresponds to the total execution time. However, we give an iterative procedure that specifies this value using several convex cost flow computations. Note that this procedure can be also used as a black box for other speed-scaling problems that can be transformed as a convex cost flow problem.

The problem. We consider the scheduling problem of minimizing the energy consumed in an open-shop setting. We are given a set of n jobs that have to be executed in a prespecified time interval $[0, d]$ on a set of m parallel processors. Each job J_j consists of m operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. Each operation $O_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq n$, has an amount of work $w_{i,j} \geq 0$ and can only be executed on the processor M_i. The preemption of the operations is allowed, while the parallel execution of operations of the same job is not permitted. A convex speed-to-power function $P(s)$ defines the energy consumption rate of any processor running at speed $s \geq 0$. The goal is to schedule the jobs within the interval $[0, d]$ so that the total energy consumption is minimized. We denote this problem as $OS|pmtn, r_j = 0, d_j = d|E$.

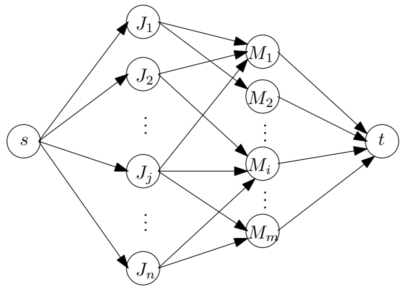
The shop scheduling problems formulate several applications in computer systems, where jobs are partitioned into tasks of different requirements which usually cannot be executed in parallel as they share common resources. As the tasks are of different type, specific purpose processors are used in order to optimize the execution. Hence, each task is preassigned to an appropriate processor. The open-shop problem describes such a situation. Note also that a formulation of a map-reduce environment as a shop scheduling problem has been recently proposed [13]. This formulation corresponds to the concurrent open-shop problem [18], in which the parallel execution of operations of the same job is allowed. Notice that map-reduce is a standard programming model widely used in data centers, where the energy consumption is one of the most important issues which has to be handled by the companies and the designers. In this direction, the map-reduce/concurrent open-shop problem in combination with the speed-scaling mechanism have been studied in [8], with objective to minimize the sum of the weighted completion times jobs under a given budget of energy. Finally, note that the results we present in this section hold also for the energy minimization concurrent open-shop problem.

Previous results. This problem is an extension of the preemptive open-shop problem $O|pmtn, r_j = 0, d_j = d|-$ [11] in the speed-scaling setting. In this problem, we are given a set of n jobs and a set of m processors. Each job J_j consists of a set of m operations, where the processing time of the operation $O_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq n$, is $p_{ij} \geq 0$. The open-shop constraint enforces that no pair of operations of a job are executed at the same time. The goal is to find a feasible schedule such that all operations are preemptively scheduled during the interval $[0, d]$ or decide that such a schedule does not exist. A polynomial algorithm for this problem can be found in [11].

Up to the best of our knowledge, no results were known for this problem in the speed-scaling setting until now.

4.1. Our Approach

As in the previous section, we first give a formulation of $OS|pmtn, r_j = 0, d_j = d|E$ as a convex cost flow problem. In order to compute the total execution time of all operations in an optimal schedule for $OS|pmtn$, $r_j = 0, d_j = d|E$, we use a search algorithm that repeatedly computes convex cost flows. Given the optimal value of the total execution time of all operations, an optimal convex cost flow gives the speeds, and hence the execution times, of the operations in an optimal schedule for $OS|pmtn, r_j = 0, d_j = d|E$. To compute a feasible schedule, we solve the corresponding instance of $O|pmtn, r_j = 0, d_j = d|-$ (see for example [11]).



	N_s	
arc	c	κ
(s, J_j)	d	0
(J_j, M_i)	$+\infty$	$f_{ij} \cdot P_{ij}(s_{ij})$
(M_i, t)	d	0

Figure 2: The flow network N_s for $OS|pmtn, r_j = 0, d_j = d|E$.

Convex cost flow formulation. We construct the flow network N_s (see Figure 2) which consists of a source node s, a destination node t, a job-node J_j, for each $1 \leq j \leq n$, and a processor-node M_i, for each $1 \leq i \leq m$. The network N_s contains an arc (s, J_j) for each job J_j, $1 \leq j \leq n$, an arc (M_i, t) for each processor M_i, $1 \leq i \leq m$, and an arc (J_j, M_i), $1 \leq j \leq n$ and $1 \leq i \leq m$ if

$w_{i,j} > 0$. The capacity of the arc (u,v) is

$$c_{u,v} = \begin{cases} d & \text{if } u = s \text{ and } v = J_j \\ +\infty & \text{if } u = J_j \text{ and } v = M_i \\ d & \text{if } u = M_i \text{ and } v = t \end{cases}$$

Assuming that flow $f_{u,v}$ passes through the arc (u,v), the cost of this arc is

$$\kappa_{u,v}(f_{u,v}) = \begin{cases} f_{J_j,M_i} \cdot P(\frac{w_j}{f_{J_j,M_i}}) & \text{if } u = J_j \text{ and } v = M_i \\ 0 & \text{otherwise} \end{cases}$$

As in the network for $S|pmtn, r_j, d_j|E$ presented in the previous section, in the network N_s for $OS|pmtn, r_j = 0, d_j = d|E$, the flow traversing the arcs corresponds to execution time. More specifically, if an amount $f_{u,v}$ of flow passes through the arc $(u,v) = (J_j, M_i)$, then $f_{u,v}$ corresponds to the execution time of operation $O_{i,j}$, $\frac{w_{i,j}}{f_{u,v}}$ corresponds to the speed of $O_{i,j}$ and $f_{u,v} \cdot P(\frac{w_{i,j}}{f_{u,v}})$ is the energy consumed for the execution of $O_{i,j}$. Furthermore, the flow passing through an edge (s, J_j) (resp. an edge (M_i, t)) represents the total execution time of the job J_j (resp. the total time that M_i operates). Hence, the total flow that leaves the source node corresponds to the total execution time of all operations. However, the total execution time of all operations in an optimal schedule for $OS|pmtn, r_j = 0, d_j = d|E$, and thus the total amount of flow that has to be sent from the source node to the destination node, cannot be easily computed as in the previous section. At the end of this section, we describe how to compute it in polynomial time.

The algorithm and its optimality. Our algorithm for $OS|pmtn, r_j = 0, d_j = d|E$ can be summarized as follows.

Algorithm 2

```
1: Construct the flow network $N_s$;
2: Determine the total execution time of all operations $\mathcal{T}$ in an optimal schedule;
3: Find a convex cost flow $\mathcal{F}$ of value $\mathcal{T}$ in $N_s$;
4: Determine the execution time of each operation;
5: Apply a polynomial-time algorithm for $O|pmtn, r_j = 0, d_j = d|-$ to find a feasible schedule;
```

In order to establish the correctness of our algorithm, we make use of the following lemma whose proof can be found in [15].

Lemma 3. [15] *An instance of the $O|pmtn, r_j = 0, d_j = d|-$ problem is feasible if and only if (i) $\sum_{i=1}^{m} p_{i,j} \leq d$ and (ii) $\sum_{j=1}^{n} p_{i,j} \leq d$.*

We will initially assume that the total execution time of all operations $\mathcal{T}$ in an optimal schedule for $OS|pmtn, r_j = 0, d_j = d|E$ can be computed in polynomial time.

Theorem 2. *Algorithm 2 finds an optimal schedule for $OS|pmtn$, $r_j = 0, d_j = d|E$.*

Proof. We first prove that there exists a feasible schedule for $OS|pmtn, r_j = 0, d_j = d|E$ of total execution time $\mathcal{T}$ if and only if there is a feasible flow $\mathcal{F}$ of value $\mathcal{T}$ in the network N_s.

Suppose that there is a feasible schedule for $OS|pmtn, r_j = 0, d_j = d|E$ of total execution time $\mathcal{T}$. Let $e_{i,j}$ be the execution time of the operation $O_{i,j}$. Hence, the speed of $O_{i,j}$ is $\frac{w_{i,j}}{e_{i,j}}$. Then, consider the flow $\mathcal{F}$ in N_s which is defined as follows:

$$f_{u,v} = \begin{cases} \sum_{i=1}^{m} e_{i,j} & \text{if } u = s \text{ and } v = J_j \\ e_{i,j} & \text{if } u = J_j \text{ and } v = M_i \\ \sum_{j=1}^{n} e_{i,j} & \text{if } u = M_i \text{ and } v = t \end{cases}$$

Recall that every operation must be executed during the interval $[0, d]$. Because of the open-shop constraint, it must hold that $\sum_{i=1}^{m} e_{i,j} \leq d$. Moreover, due to the fact that each processor can execute at most one operation at each time, we have that $\sum_{j=1}^{n} e_{i,j} \leq d$. Therefore, the flow $\mathcal{F}$ is of value $\mathcal{T}$ and it is a feasible flow in the network N_s.

To the other direction, assume that there exists a feasible flow $\mathcal{F}$ of value $\mathcal{T}$ in the network N_s. We can then define a feasible schedule for $OS|pmtn, r_j = 0, d_j = d|E$ by setting the execution time of the operation $O_{i,j}$ to be equal to f_{J_j,I_i}, i.e., we set the speed of $O_{i,j}$ equal to $\frac{w_{i,j}}{f_{J_j,I_i}}$. Since $\mathcal{F}$ is feasible, it must hold that $\sum_{i=1}^{m} f_{J_j,I_i} \leq d$ and $\sum_{j=1}^{n} f_{J_j,I_i} \leq d$. Using Lemma 3, we can construct a feasible schedule for $OS|pmtn, r_j = 0, d_j = d|E$.

We conclude the proof with the optimality of our algorithm. Among the set of feasible flows of value $\mathcal{T}$, the algorithm finds the one which minimizes the term $\sum_{i=1}^{m}\sum_{j=1}^{n} f_{J_j,M_i} \cdot P(\frac{w_j}{f_{J_j,M_i}})$. In other words, given our convex cost flow construction, the algorithm finds the schedule with the minimum energy among the schedules of total execution time $\mathcal{T}$. But, we have assumed that there exists an optimal schedule of total execution time equal to $\mathcal{T}$. Hence, our algorithm is optimal. $\square$

4.2. Computing the Total Execution Time

It remains to show how we algorithmically determine the total execution time of all operations in an optimal schedule for $OS|pmtn, r_j = 0, d_j = d|E$.

First, we introduce some additional notation. Henceforth, we denote by T^* the sum of execution times of all operations in an optimal schedule for $OS|pmtn$, $r_j = 0, d_j = d|E$. Let $\mathcal{S}$ be any feasible schedule for the problem and assume that $t_{i,j}$ is the execution time of the operation $O_{i,j}$ in $\mathcal{S}$, $1 \leq i \leq m$ and $1 \leq j \leq n$. We denote by $\vec{t} = (t_{1,1}, t_{2,1}, \ldots, t_{m,1}, t_{1,2}, \ldots, t_{m,n})$ the vector that contains the execution times of all the operations. Then, let $T(\vec{t}) = \sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij}$ and $E(\vec{t}) = \sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij} \cdot P(\frac{w_{ij}}{t_{ij}})$ be the functions that map any vector of execution times $\vec{t}$ to the total execution time and the total energy consumption of the schedule $\mathcal{S}$. Note that, $E(\vec{t})$ is convex with respect to the vector $\vec{t}$ as a sum of convex functions. Furthermore, we define the function $E^*(T) = \min\{E(\vec{t}) : T(\vec{t}) = T\}$ which indicates the minimum energy consumption when the sum of execution times of all operations has to be equal to T.

Proposition 1. *$E^*(T)$ is convex with respect to T.*

Proof. Consider three values of total execution times $T_1, T_2, T_3 > 0$, and let $\vec{t_1}$, $\vec{t_2}$, $\vec{t_3}$ be three corresponding optimal vectors of execution times, respectively. That is, (i) $T(\vec{t_1}) = T_1$, $T(\vec{t_2}) = T_2$, $T(\vec{t_3}) = T_3$, and (ii) $E^*(T_1) = E(\vec{t_1})$, $E^*(T_2) = E(\vec{t_2})$, $E^*(T_3) = E(\vec{t_3})$. In other words, $\vec{t_1}$, $\vec{t_2}$, $\vec{t_3}$ define optimal schedules (with respect to minimizing the total energy consumption) given that the total execution time of all jobs must be equal to T_1, T_2, T_3, respectively. Without loss of generality, assume that $T_1 \leq T_3 \leq T_2$. Clearly, there must be a $\theta \in [0,1]$ such that $T_3 = \theta T_1 + (1-\theta)T_2$. Consider, now, the vector $\vec{t} = \theta\vec{t_1} + (1-\theta)\vec{t_2}$. As $T(\vec{t})$ is linear with respect to $\vec{t}$, it holds that

$$T(\vec{t}) = T(\theta\vec{t_1} + (1-\theta)\vec{t_2}) = \theta T(\vec{t_1}) + (1-\theta)T(\vec{t_2}) = T_3 = T(\vec{t_3})$$

By the definition of $E^*(T)$, it holds that $E^*(T_3) \leq E(\vec{t})$. Moreover, recall that the function $E(\vec{t})$ is convex with respect to $\vec{t}$. In all, we have that

$$\begin{aligned} E^*(\theta T_1 + (1-\theta)T_2) \;&=\; E^*(T_3) \;\leq\; E(\vec{t}) \;=\; E\left(\theta\vec{t_1} + (1-\theta)\vec{t_2}\right) \\ &\leq\; \theta E(\vec{t_1}) + (1-\theta)E(\vec{t_2}) \;=\; \theta E^*(T_1) + (1-\theta)E^*(T_2) \end{aligned}$$

and hence, the function $E^*(T)$ is convex with respect to T. □

Next, we give the search algorithm that finds the value $T^* = \arg\min_T\{E^*(T)\}$ with accuracy $1/\varepsilon$. Consider any $T_1, T_2, T_3 > 0$ such that $T_1 < T_2 = \frac{T_1+T_3}{2} < T_3$. As $E^*(T)$ is convex, we have that $E^*(T_2) \leq \frac{E^*(T_1)+E^*(T_3)}{2}$. Therefore, it follows that either $E^*(T_2) \leq E^*(T_1)$ or $E^*(T_2) \leq E^*(T_3)$ (or both). If only the first is true, then we reduce our search space to $[T_2, T_3]$. Accordingly, if only the second is true, then we reduce our search space to $[T_1, T_2]$. Finally, if both are true, then we reduce our search space to one of the following intervals: $[T_1, T_2]$, $[T_2, T_3]$ or $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$. If $E^*(\frac{T_1+T_2}{2}) \leq E^*(T_2)$, then the search space is reduced to $[T_1, T_2]$. If $E^*(\frac{T_2+T_3}{2}) \leq E^*(T_2)$, then the search space is reduced to $[T_2, T_3]$. Finally, if $E^*(\frac{T_1+T_2}{2}) > E^*(T_2)$ and $E^*(\frac{T_2+T_3}{2}) > E^*(T_2)$, then the search space is reduced to $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$. The correctness of all the cases is based on the fact that $E^*(T)$ is convex. We call this procedure Algorithm FIND-FLOW and we initialize T_1, T_2 and T_3 with 0, $\frac{\mathcal{T}}{2}$ and $\mathcal{T}$, respectively, where $\mathcal{T}$ is an upper bound on the sum of execution times for all operations, i.e., $\mathcal{T} = m \cdot d$.

Lemma 4. *Algorithm* FIND-FLOW *returns a value T^* such that the term $E^*(T^*)$ is minimized among all $T^* > 0$. The complexity of the algorithm is $O(nm\log(md)(nm + (n+m)\log(n+m))(\log\frac{md}{\varepsilon}))$, where ε is the accuracy of the machine.*

Algorithm FIND-FLOW

```
1: Initialize: $T_1 = 0$, $T_2 = \frac{\mathcal{T}}{2}$, $T_3 = \mathcal{T}$;
2: while $T_3 - T_1 > \varepsilon$ do
3:   Compute $E^*(T_1)$, $E^*(T_2)$ and $E^*(T_3)$ by finding a convex cost flow of
     value $T_1$, $T_2$ and $T_3$, respectively, in the network $N_s$;
4:   if $E^*(T_3) < E^*(T_2) \leq E^*(T_1)$ then
5:     Update: $T_1 = T_2$ and $T_2 = \frac{T_1+T_3}{2}$;
6:   if $E^*(T_1) < E^*(T_2) \leq E^*(T_3)$ then
7:     Update: $T_3 = T_2$ and $T_2 = \frac{T_1+T_3}{2}$;
8:   if $E^*(T_2) \leq E^*(T_1)$ and $E^*(T_2) \leq E^*(T_3)$ then
9:     Compute $E^*(\frac{T_1+T_2}{2})$ and $E^*(\frac{T_2+T_3}{2})$ by finding a convex cost flow of
       value $\frac{T_1+T_2}{2}$ and $\frac{T_2+T_3}{2}$, respectively, in the network $N_s$;
10:    if $E^*(\frac{T_1+T_2}{2}) < E^*(T_2)$ then
11:      Update: $T_3 = T_2$ and $T_2 = \frac{T_1+T_3}{2}$;
12:    if $E^*(\frac{T_2+T_3}{2}) < E^*(T_2)$ then
13:      Update: $T_1 = T_2$ and $T_2 = \frac{T_1+T_3}{2}$;
14:    if $E^*(\frac{T_1+T_2}{2}) \geq E^*(T_2)$ and $E^*(\frac{T_2+T_3}{2}) \geq E^*(T_2)$ then
15:      Update: $T_1 = \frac{T_1+T_2}{2}$ and $T_3 = \frac{T_2+T_3}{2}$;
16: return $T_2$;
```

Proof. At each iteration of the algorithm, the search space is reduced. In a given iteration, this reduction is accomplished by removing one of the intervals $[T_1, T_2]$, $[T_2, T_3]$, or $[T_1, \frac{T_1+T_2}{2}] \cup [\frac{T_2+T_3}{2}, T_3]$. In order to establish the correctness of our algorithm, it suffices to show that, at the end of each iteration, there is a value T^* in the algorithm's remaining search space that minimizes the term $E^*(T^*)$.

Suppose that $[T_2, T_3]$ is the interval removed by the algorithm and assume for the sake of contradiction that $T^* \in [T_2, T_3]$, for any $T^* = \arg\min_T\{E^*(T)\}$. Since the interval $[T_2, T_3]$ is removed by the algorithm, we have that either (i) $E^*(T_1) < E^*(T_2) \leq E^*(T_3)$, or (ii) $E^*(T_2) \leq E^*(T_1)$, $E^*(T_2) \leq E^*(T_3)$ and $E^*(\frac{T_1+T_2}{2}) < E^*(T_2)$.

Consider first the case (i). Let T^* be the total execution time of an optimal solution. Since $T^* \in [T_2, T_3]$, we know that there is a $\theta \in [0, 1]$ such that $T_2 = \theta T_1 + (1-\theta)T^*$. Then, due to the convexity of the function $E^*(T)$ we have that $E^*(T_2) = \theta E^*(T_1) + (1-\theta)E^*(T^*)$. By the way we picked T^*, it holds that $E^*(T^*) \leq E^*(T_1)$. Hence, we get that $E^*(T_2) \leq E^*(T_1)$, which is a contradiction.

In the same vein, consider now the case (ii). Assume that $T^* \in [T_2, T_3]$. Then, there is a $\theta \in [0, 1]$ such that $T_2 = \theta(\frac{T_1+T_2}{2}) + (1-\theta)T^*$. By the convexity of the $E^*(T)$ function, we get as before that $E^*(T_2) \leq E^*(\frac{T_1+T_2}{2})$, which is a contradiction.

Note that, if the interval ignored by the algorithm is $[T_1, T_2]$, then the fact that $T^* \in [T_2, T_3]$ can be proved with almost the same manner as in the case where the interval $[T_2, T_3]$ is removed.

Finally, suppose that the algorithm reduces the search space to $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$ and assume for contradiction that $T^* \in [\frac{T_2+T_3}{2}, T_3]$ (note that it might also be the case that $T^* \in [T_1, \frac{T_1+T_2}{2}]$; this case can be handled analogously). Clearly, it holds that $E^*(\frac{T_2+T_3}{2}) \leq \frac{E^*(T_2)+E^*(T_3)}{2} \leq E^*(T_3)$. That is, we have that $T_2 \leq \frac{T_2+T_3}{2} \leq T^* \leq T_3$ and $E^*(T_2), E^*(\frac{T_2+T_3}{2}), \leq E^*(T_3)$. Hence, we can reach a contradiction as before.

In order to compute the complexity of the algorithm, note that the network constructed by the algorithm contains $O(n+m)$ nodes and $O(nm)$ edges. The maximum capacity of an edge is d while the total flow that crosses the network can be at most md. Thus, the overall complexity of finding a convex cost flow in the network is $O(nm\log(md)(nm+(n+m)\log(n+m)))$. Note now that in each iteration the search domain, i.e., the interval $[T_1, T_3]$, reduces by half. In each iteration at most five convex cost flows on network N_s are sought. Therefore, for the complexity of our algorithm we have the following recursion:

$$\begin{gathered} F(B) = F(B/2) + 5 \cdot O(nm\log(md)(nm + (n+m)\log(n+m))) \\ F(\varepsilon) = 1 \end{gathered}$$

Solving this recurrence for $B = \mathcal{T} = m \cdot d$, the lemma follows. $\square$

Note that, in order to compute the total execution time T^* of all the operations in an optimal schedule, the algorithm performs a binary search in the interval $[0, m \cdot d]$. However, it has to be noticed that in machines with limited precision, accuracy issues might occur. More specifically, our algorithm produces an $OPT + \varepsilon$ solution in time polynomial to $\log \frac{m \cdot d}{\varepsilon}$, where ε is the accuracy of the machine. In other words, the complexity of our algorithm is polynomial not only to the dimensions of the problem (i.e., n and m) but also to the number of bits needed to represent T^*, that is $\log \frac{m \cdot d}{\varepsilon}$. This kind of algorithms are usually called "weakly polynomial" algorithms[2].

5. Mean Completion Time plus Energy Minimization on Multiprocessors without Preemptions

The problem. We consider the multiprocessor scheduling problem of minimizing a linear combination of the sum of completion times of a set of n jobs and their total energy consumption. The jobs have to be executed by a set of m parallel processors where the preemption and migration of jobs are not allowed. Each job J_j, $1 \leq j \leq n$, has an amount of work w_j to accomplish and all jobs are released at time $t = 0$. We denote by C_j the completion time of job J_j, $1 \leq j \leq n$. A convex speed-to-power function $P(s)$ defines the energy consumption rate when a job is executed with speed $s > 0$ on any processor. The goal is to minimize the sum of completion times of all the jobs plus β

[2]For a discussion about weakly and strongly polynomial algorithms, see for example the introduction in [1].

times their total energy consumption. The parameter $\beta > 0$ is used to specify the relevant importance of the mean completion time criterion versus the total energy consumption criterion. We denote this problem as $S||\sum C_j + \beta \cdot E$.

Known results. This problem is an extension in the speed-scaling setting of the problem $P||\sum C_j$ of scheduling non-preemptively a set of n jobs, each one characterized by its processing time p_j, on a set of m machines such that the sum of the completion times of all jobs is minimized. A polynomial-time algorithm has been proposed for $P||\sum C_j$ [12].

The single-processor speed-scaling problem without preemptions of minimizing the jobs' mean completion time has been studied by Albers et al. [6] and Pruhs et al. [20] in the presence of release dates and unit work jobs. In [20] the objective is the minimization of the sum of the flow times of the jobs[3] under a given budget of energy, while in [6] the goal is to minimize the sum of the flow times of the jobs and of the consumed energy.

5.1. Our Approach

The main idea is to formulate $S||\sum C_j + \beta \cdot E$ as a problem of searching for a minimum weighted maximum matching in an appropriate bipartite graph. This formulation is based on two observations. Firstly, the fact that the preemption and the migration of jobs is not allowed means that there is an order of the jobs executed by any processor in any feasible schedule. Given such a schedule, if ℓ jobs are executed by the processor M_i, then we can consider that there are ℓ available positions on M_i, one for the execution of each of the ℓ jobs. If the job J_j is executed in the k-th position of the processor M_i, then $k-1$ jobs precede J_j and $\ell - k$ jobs succeed J_j. Clearly, there can be at most n such positions for each processor. Secondly, the contribution of a job J_j to the objective function depends only on its position on the processor on which it is executed and it is independent of where the other jobs are executed. Overall, our problem reduces to assigning every job to a position of a machine so that our objective is minimized.

Minimum weighted maximum matching formulation. In order to formulate the $S||\sum C_j + \beta \cdot E$ problem as a minimum weighted maximum matching problem, we define a bipartite graph G_s whose edges are weighted. The following lemma is our guide for assigning weights on the edges of G_s and fixes the cost of executing a job J_j to the k-th position of any processor.

Lemma 5. *Assume that in an optimal schedule for $S||\sum C_j + \beta \cdot E$ the job J_j, $1 \leq j \leq n$, is executed at speed s_j on processor M_i in the k-th position from the end of M_i. Then, the total contribution of J_j to the objective function is minimized if it holds that $s_j P'(s_j) - P(s_j) = \frac{k}{\beta}$, where $P'(s)$ is the derivative of the power function P with respect to the speed s.*

[3]The flow time of a job is defined as the amount of time that the job spends in the system, i.e., the difference between its completion time and its release date.

Proof. The execution time of job J_j is $\frac{w_j}{s_j}$. Since $k-1$ jobs follow J_j on M_i, the term $\frac{w_j}{s_j}$ is added k times on the sum of completion times of all the jobs. Moreover, $\frac{w_j}{s_j}P(s_j)$ units of energy are consumed for the execution of the job J_j. Hence, the total contribution of J_j to the objective is $k \cdot \frac{w_j}{s_j} + \beta \cdot \frac{w_j}{s_j}P(s_j)$. By differentiating the last term with respect to s_j and setting this derivative equal to zero, we can get the value of s_j for which this contribution is minimized. $\square$

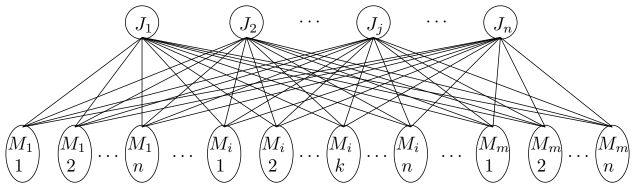


Figure 3: The bipartite graph G_s for $S||\sum C_j + \beta \cdot E$. Each edge $(J_j, (M_i, k))$ has a weight $\kappa_{i,j,k} = k \cdot \frac{w_j}{s_j} + \beta \cdot \frac{w_j}{s_j}P(s_j)$.

Based on the above lemma, we create the complete bipartite graph $G_s = (V, U; E)$ as follows: (i) for each job J_j, $1 \leq j \leq n$, we add a vertex in V, (ii) for each processor M_i, $1 \leq i \leq m$, and each position k, $1 \leq k \leq n$, (counting from the end) we add a vertex in U, and (iii) for each edge $(J_j, (M_i, k))$, we set its weight $\kappa_{i,j,k} = k \cdot \frac{w_j}{s_j} + \beta \cdot \frac{w_j}{s_j}P(s_j)$ where s_j is computed according to Lemma 5.

The algorithm and its optimality. Recall that each job J_j runs at a constant speed s_j in any optimal schedule for $S||\sum C_j + \beta \cdot E$. Moreover, based on a similar argument, it holds that there is no idle period on any processor between the common release date of the jobs and the date at which the last job completes its execution, in any optimal schedule. A description of our algorithm follows.

Algorithm 3

```
1: Construct the bipartite graph $G_s$;
2: Find a minimum weighted maximum matching $M$ in $G_s$;
3: for each $(J_j, (M_i, k)) \in M$ do
4:     Schedule $J_j$ to the position $k$ of $M_i$ with speed $s_j$ such that
       $s_j P'(s_j) - P(s_j) = \frac{k}{\beta}$;
```

Theorem 3. *Algorithm 3 finds an optimal schedule for $S||\sum C_j + \beta \cdot E$ in $O(n^2 m^2 (n + \log m))$ time.*

Proof. By the construction of G_s, the vertex J_j, $1 \leq j \leq n$, can belong to at most one edge of the matching. Moreover, the number of the job nodes is less than the number of the processor-position nodes and every job node is connected with every processor-position node. Hence, every job node belongs to a maximum matching of G_s. Therefore, each job is scheduled exactly once in a single processor and, as a result, the schedule produced by the algorithm is feasible.

We next prove the optimality of our algorithm. Among the matchings with cardinality n, the algorithm finds the one which minimizes the term $\sum_{j=1}^{n}(k \cdot \frac{w_j}{s_j} + \beta \cdot \frac{w_j}{s_j} P(s_j))$, where s_j has been selected in an optimal way according to Lemma 5. In other words, given the construction of the bipartite graph G_s, the algorithm finds the schedule with the minimum energy consumption. Hence, our algorithm is optimal.

For the complexity of our algorithm, observe first that our graph contains $O(nm)$ nodes and $O(n^2 m)$ edges. Therefore, the overall complexity of our algorithm is $O(n^3 m^2 + n^2 m^2 \log m)$. $\square$

6. More Problems and Conclusions

The idea of formulating a problem as a convex cost flow problem can be applied in order to solve other scheduling problems in the speed-scaling setting as well. As examples, we briefly discuss here the following problems:

(a) *Preemptive speed-scaling scheduling malleable jobs without migration costs.* In this problem, a job can be executed by more than one processor in parallel, decreasing in this way its total execution time. If each job can use any number of processors, then the problem reduces to the single processor case. We consider here the case where each job j is allowed to use at most $\delta_j \leq m$ processors at the same time (see for example [14]). Note that, in some applications, this parallelization may add an additional cost to the execution time of a job due to communication issues. Here, we consider that we pay no cost for this. A convex cost flow formulation for this problem is almost the same with the one where the jobs are not malleable. The main difference is that the total execution time of a job during an interval I_i is upper bounded by $\delta_j |I_i|$ since it is allowed to be executed by at most δ_j processors at the same time.

(b) *Restricted (or multi-purpose) multiprocessor speed-scaling problem with migrations.* This problem is a generalization of the multiprocessor problem with migrations in which every job can be executed only by a subset of the available processors. More specifically, each job is associated with a subset of processors, and it can be executed only on one of the processors of its set at each time. We can model this problem as a convex cost flow problem in a more complicated flow network. This flow network is similar to the network proposed in [11] (page 280) for the feasibility version of the same problem when speed-scaling is not permitted, i.e., the energy consumption issues are not considered.

In order to determine the value of the flow that crosses the network in the above problems, we can use the general searching procedure proposed in Section 4.2.

A reasonable question that arises after all is whether we can characterize the problems that can be solved using the convex cost flow transformation.

A common characteristic of these problems up to now is that their feasibility question when speed-scaling is not allowed can be answered through a maximum flow network. However, the opposite does not seem to be always true. Finally, we would like to mention as a very interesting open question the complexity of the single-processor speed scaling problem with preemptions and release dates when the objective is to minimize the sum of the completion times of the jobs plus the total energy consumption.

Acknowledgements

E. Bampis, D. Letsios and G. Lucarelli are partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, by the project ALGONOW, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES, by a French-Chinese Cai Yuanpei project, and by the project "Mathematical Programming and Non-linear Combinatorial Optimization" under the program PGMO.

[1] I. Adler, C. Papadimitriou, and A. Rubinstein. On simplex pivoting rules and complexity theory. In *17th International Conference on Integer Programming and Combinatorial Optimization*, volume 8494 of *LNCS*, pages 13–24. Springer, 2014.

[2] R. K. Ahuja and T. L. Magnanti J. B. Orlin. *Network flows: theory, algorithms and applications.* Prentice Hall, 1993.

[3] S. Albers. Energy-efficient algorithms. *Commun. ACM*, 53:86–96, 2010.

[4] S. Albers. Algorithms for dynamic speed scaling. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 1–11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[5] S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration: extended abstract. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 279–288. ACM, 2011.

[6] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. on Algorithms*, 3(4):Article 49, 2007.

[7] E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *18th International European Conference on Parallel and Distributed Computing (Euro-Par)*, volume 7484 of *LNCS*, pages 128–140. Springer, 2012.

[8] E. Bampis, V. Chau, D. Letsios, G. Lucarelli, I. Milis, and G. Zois. Energy efficient scheduling of mapreduce jobs. In *18th International European Conference on Parallel and Distributed Computing (Euro-Par)*, volume 8632 of *LNCS*, pages 198–209. Springer, 2014.

[9] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 693–701, 2009.

[10] B. D. Bingham and M. R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 153–161. IEEE, 2008.

[11] P. Brucker. *Scheduling algorithms (4th ed.)*. Springer, 2004.

[12] J. Bruno, Jr. E.G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17:382–387, 1974.

[13] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *30th IEEE International Conference on Computer Communications (INFOCOM)*, pages 3074–3082. IEEE, 2011.

[14] M. Drozdowski. *Scheduling parallel tasks - Algorithms and complexity*, chapter 25. Handbook of scheduling: Algorithms, models and performance analysis. Chapman & Hall/CRC, 2004.

[15] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, 1976.

[16] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[17] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 256–264, 2008.

[18] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38:390–395, 2010.

[19] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[20] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *ACM Trans. on Algorithms*, 4(3):Article 38, 2008.

[21] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.