Time-Constrained Energy Minimization for Online Execution of a Stochastic DAG Task

Jonatha Anselmi, Bruno Gaujal, Karl Gottlieb

Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, Grenoble, 38000, France.

Contributing authors: jonatha.anselmi@inria.fr; bruno.gaujal@inria.fr; karl.gottlieb@inria.fr;

Abstract

We study the problem of energy-efficient online execution of a complex task on a server with variable processing speed. The task consists of a set of stochastic elementary jobs structured as a Directed Acyclic Graph (DAG), where each job's execution may reveal new information that influences future scheduling decisions. Our objective is to determine an online speed control policy that minimizes the expected energy consumption while ensuring that the task completes before a strict deadline. Leveraging tools from convex optimization, the optimality principle, and backward induction, we derive a structural characterization of the optimal policy. We find that this is linked to a set of second-order differential equations and exhibits a non-trivial form. Building on this result, we develop a discretization-based algorithm that efficiently approximates the optimal policy. The proposed algorithm is provably asymptotically exact in the discretization step and has computational complexity $\boldsymbol{O(AN)}$, where $\boldsymbol{A}$ denotes the number of edges in the underlying DAG and $\boldsymbol{N}$ is the size of the discretization. Our results offer a principled and computationally efficient solution framework for online execution of structured stochastic workloads under strict energy and timing constraints.

Keywords: Energy minimization, online scheduling, real time, multiple jobs, speed profile.

1 Introduction

Modern computing systems increasingly rely on dynamic, energy-efficient resource management strategies to handle growing workloads under strict performance constraints. Whether in embedded real-time systems, large-scale cloud infrastructures, or high-performance GPU clusters, energy consumption has become a critical bottleneck – impacting both operational costs and environmental sustainability [1]. In parallel, the complexity of workloads has increased, with tasks frequently comprising multiple interdependent stages that must be executed under tight timing constraints [2]. This is particularly true for deep learning training tasks, which have a pervasive impact across various domains of daily life [3].

In this work, we study the online speed scheduling of a complex task composed of several interconnected jobs on a processing unit with variable speed capabilities. Our objective is to determine a speed policy that minimizes the expected energy consumption while ensuring that the task completes before a fixed, hard deadline. This class of problems has been extensively studied in both applied and theoretical literature; see, e.g., [2, 4–6]. The novelty of our approach lies in the combination of three key aspects: (i) job sizes are stochastic and unknown in advance, (ii) the scheduler may adapt its speed strategy based on online observations revealed during task execution, e.g., the exact size of each completed job, and (iii) the task exhibits a general Directed Acyclic Graph (DAG) structure, reflecting real-world job dependencies.

1.1 Motivation and Practical Relevance

Energy matters. It is increasingly recognized that the energy consumed by computer systems – including data centers, personal devices, and network infrastructure – represents a substantial and growing share of global electricity usage [7]. Recent estimates indicate that information and communication technologies (ICT) account for approximately 4% of global electricity consumption, a figure projected to rise significantly as digital services expand and artificial intelligence workloads grow [8]. In large-scale data centers powering artificial intelligence training pipelines or scientific simulations, power consumption is often the dominant cost factor. For instance, training a modern transformer-based model can consume hundreds of megawatt-hours of electricity [1]. In mobile or embedded systems, high energy consumption leads to rapid battery drain or heat dissipation issues, which directly impact usability and safety. For these reasons, it is imperative to design energy-aware execution strategies that exploit all available flexibility, including variable-speed processing and runtime information.

Why online scheduling? In many real-world systems, job characteristics such as execution time or size are not known in advance. This uncertainty arises naturally in diverse contexts: data processing pipelines with unpredictable input sizes, compute-intensive workloads with stochastic computational paths, or dynamically generated sub-tasks in serverless and edge computing environments. As such, the online setting, in which the scheduler only gains information about a job's size once it completes, is both realistic and necessary [9–11]. Traditional offline approaches that assume full

knowledge of the workload are inadequate in these scenarios, as they do not fully exploit the information revealed in real time.

Why DAG-structured tasks? A distinctive feature of our model is that the task is composed of multiple elementary jobs, interconnected via a DAG. This structure accurately mirrors many modern computing workloads. For example, machine learning pipelines (a particular case of DAG) involve stages such as data preprocessing, model loading, training, and evaluation, where some stages run in parallel and others depend on intermediate outputs [3]. DAG scheduling frameworks are therefore fundamental in distributed computing settings [9]. In such workflows, *checkpointing* plays a key role: it allows the system to record progress after completing each stage, revealing the actual amount of work completed and enabling the scheduler to adjust the strategy for the remaining tasks. This naturally provides online information and justifies the use of a DAG model, where the flow of execution is governed by probabilistic transitions between stages. DAG-based task representations are also pervasive in serverless computing, where event-driven function executions can spawn follow-up tasks based on branching logic or workflow rules.

1.2 Contribution

Given a task defined by a DAG of stochastic jobs and a hard deadline D, we look for an *online* speed profile that minimizes the expected energy consumption. The processing unit supports continuous speed scaling, and the energy cost follows a standard power function model of the speed. The scheduler knows the job dependencies and their probability distributions, but not the actual sizes until completion. Thus, it must dynamically adapt the speed based on revealed information, making this an instance of stochastic optimal control with partial observability.

Our central questions are: What is the structure of the optimal speed policy? Can it be computed efficiently? How does the task's DAG structure influence the strategy? This paper addresses these questions by deriving the structure of the optimal online policy and proposing an efficient algorithm to compute it. Specifically, our results can be summarized as follows:

- We characterize the optimal online speed policy for DAG-structured tasks under stochastic job sizes and hard deadlines. We show that the optimal speed profile is strictly related to the solution of a set of second-order differential equations, one for each job of the task, whose solution in "closed form" is out of reach. This is progressively established in Theorems 1, 2, 4 and 5.
- Leveraging the structural properties of the optimal policy established in our results, we then develop a discretization-based algorithm to compute an approximate solution to the continuous optimal speed profile and prove in Theorem 3 that its output converges indeed to the optimal speed profile in the limit where the granularity of the discretization grows to infinity.
- In the general DAG setting, the proposed algorithm is efficient as we show that its computational complexity $O(AN)$, where A is the number of edges and N is the discretization size.

To improve the readability of our work, we develop our results starting with the case of a task with two sequential jobs, as this already captures the key technical difficulties of the considered mathematical problem. Then, we progressively extend our approach to K sequential jobs and, finally, to the general DAG setting.

Our results rely on a suite of mathematical techniques including convex analysis (for uniqueness and construction of the optimal speed policy), backward induction (to recursively solve subproblems along the DAG structure), and discretization (to develop practical numerical algorithms).

To the best of our knowledge, existing methods do not provide optimality guarantees or computationally efficient solutions as we do. This is the first work to provide a complete theoretical and algorithmic solution to the online energy-aware scheduling of DAG-structured stochastic tasks with hard deadlines. We point the reader to Section 2 for a high-level review of the related literature and to Section 4 for a detailed review of the specific mathematical problem considered in this paper.

1.3 Organization

The remainder of the paper is structured as follows. Section 2 presents a review of the related literature, highlighting both classical results and recent advances in energy-aware scheduling. Section 3 introduces the formal framework for energy minimization under hard deadline constraints and defines the mathematical problem of interest. In Section 4, we delve into a reasoned review of the existing approaches, illustrating their limitations and emphasizing the technical subtleties of speed scheduling for multiple stochastic jobs. Section 5 develops the optimal solution for the case where the task consists of two sequential jobs, which serves as a foundational stepping stone. Building on these insights, Section 6 extends the results to the case of tasks with arbitrary DAG structure. Finally, Section 7 concludes the paper and outlines directions for future research.

2 Related Work

The problem of minimizing energy consumption under timing constraints has been extensively studied across a variety of scheduling frameworks [2, 4, 9]. Key distinctions in the literature arise from the assumptions on the scheduler's information (offline vs. online) and the characteristics of the processing unit (continuous vs. discrete speed, presence of ramp-up costs, etc.). In this section, we review the most relevant contributions and highlight how our work differs in terms of both modeling assumptions and theoretical guarantees.

Offline Scheduling.

Early work in energy-aware scheduling focused on the *offline* setting, where job characteristics such as release times, deadlines, and processing requirements are known a priori. A foundational contribution in this domain is the work of Yao et al. [12], which introduced a framework for minimizing energy consumption using dynamic voltage and frequency scaling (DVFS). Subsequent works [10, 13, 14] refined the algorithmic

complexity and addressed increasingly general task models. In these studies, the problem reduces to a deterministic convex optimization problem over known quantities, allowing for efficient computation of optimal speed profiles.

Online Scheduling.

In contrast, we focus on the more challenging *online* setting, where job sizes are not known in advance and are only revealed upon completion. This reflects a more realistic model for modern computing systems that must operate under uncertainty. One of the earliest results in this direction is due to Lorch and Smith [5], who considered the execution of a single stochastic job under an ideal processor model with continuous speeds and no transition costs. Under these assumptions, they derived a closed-form expression for the optimal speed profile that depends only on the job's size distribution and the deadline. The optimal speed profile developed in [5] has been recently extended in [15] to the case where the maximum processing speed is bounded and the power function is "general".

In contrast to continuous-speed models, the work in [6] investigates *discontinuous* speed profiles, which naturally arise in multi-core or multi-GPU architectures where only a finite set of processing speeds is available. Under mild conditions on the power consumption function, the authors show that the optimal speed profile can be characterized as the unique solution to a strictly convex optimization problem. They further propose an ad-hoc algorithm that solves this problem efficiently, with logarithmic complexity $O(\ln n)$ in the number of available speeds n.

Markov Decision Process Formulations.

The online scheduling problem with stochastic jobs and hard or soft deadlines has also been formulated and studied using the theory of Markov Decision Processes (MDPs). In [16] and [17], the authors explored MDP-based formulations for optimizing energy consumption under both hard and soft deadline constraints. These works consider sequential decision-making under uncertainty and provide optimal policies using dynamic programming techniques. Our approach here is different because it considers the multi-job setting, where the task consists of a sequence of stochastic jobs whose structure is captured by a DAG. We maintain the hard deadline constraint and assume that the scheduler only has access to the job size distributions, not their realizations until completion. We also adopt the idealized continuous-speed processor model used in [5] and [17].

Multi-Job Scheduling.

Although some prior works have addressed the case of multiple jobs, they often rely on strong simplifying assumptions or offer suboptimal solutions. A common approach for determining the speed profile in the multi-job setting is to develop heuristics that leverage structural properties derived from the single-job case, as explored in [18]. In [19], the authors assume a fixed speed for each job, leading to inefficient execution strategies, as optimality generally requires adjusting the speed continuously during job execution. In [20], a more flexible approach is proposed: each job is assigned a fixed sub-deadline, and optimal speed profiles are computed individually. However,

their method is based on an unproven claim of optimality and, as we show later, may lead to suboptimal energy consumption, since the policy fails to adapt to information revealed during task execution.

Positioning of our Work.

In contrast to the works above, our study provides a complete and rigorous solution to the problem of determining the optimal online speed profile for the execution of a stochastic DAG-structured task under a hard deadline. We characterize the structure of the optimal speed profile, derive an efficient algorithm based on discretized dynamic programming, and prove that it is exact in the limit where the discretization size goes to infinity. To the best of our knowledge, this is the first work to solve the general multi-job online scheduling problem with both theoretical guarantees and practical efficiency.

3 Energy-Optimization Framework

We consider an energy optimization framework consisting of three key components: a processing unit (or server), a task composed of several elementary jobs, and a scheduler. The processing unit is responsible for executing the task, and the scheduler decides *online* at what speed the task must be processed at any point in time. The objective is for the server to determine an online speed profile that minimizes the expected energy consumption while ensuring that the task is completed before a given strict deadline. In the following, we describe these components more precisely and formally define the mathematical problem investigated in this paper.

3.1 Processing Unit

The processing unit operates at an adjustable speed denoted by s operations per time unit. In practice, this unit can represent a system utilizing the Dynamic Voltage and Frequency Scaling (DVFS) technology, a multi-GPU system as in [6], or a more advanced computing infrastructure with speed scaling capabilities. We assume that the set of permissible speeds is $\mathbb{R}_+$, which means that the maximal speed is unbounded. Following the same approach as in [15, Theorem 1], our results can be generalized to the case where the processing speed is bounded. However, we omit this generalization to avoid unnecessarily complicating both the presentation and the proofs.

The processing unit consumes electric energy when active. We assume that the power consumption of the processing unit when operating at speed $s \in \mathbb{R}_+$ has the form $P(s) = \beta\, s^\alpha$, where $\alpha \geq 2$ and $\beta > 0$ is a constant. This model is classical in the literature; see, e.g., [5, 6, 13, 21]. In the rest of this paper, we fix $\beta = 1$ for simplicity, all mean energy terms can be multiplied by β to go back to the original case.

3.2 Task

The processing unit executes a task composed of K elementary jobs, referred to as $J_1, \ldots, J_K$. We begin by defining what constitutes a job, and then we build on this definition to formally define a task.

A job is a sequence of non-preemptive elementary operations whose *size*, or equivalently amount of *work*, is represented by the random variable W_k, for job J_k. The realization of W_k is unknown beforehand. However, its statistical distribution is known a priori and given by the cumulative distribution function $F_k(w) := \mathbb{P}(W_k \leq w)$. This describes the probability that the size of job k does not exceed w. We also define its complementary cumulative distribution function as $F_k^c(w) := \mathbb{P}(W_k > w)$. The size of job J_k is assumed to belong to the interval $[W_k^{\min}, W_k^{\max}]$, where $W_k^{\min} \geq 0$ and $W_k^{\max} < \infty$ represent its minimum and maximum possible sizes. All the job sizes W_k are assumed independent.

Given the above job description, a task is defined as a set of K jobs whose execution is connected according to a flowchart structure. Specifically, the execution of the given task begins with job J_i with probability p_{0i}. Upon completion of job J_i, the task execution terminates with probability p_{i0} or proceeds to job J_j with probability p_{ij}. In the latter case, after completion of job J_j, the task execution terminates with probability p_{j0} or proceeds to job J_k with probability p_{jk}, and so on. We let the transition probability p_{ij} from job J_i to job J_j depend on the (random) size W_i of job J_i. Thus, $p_{ij} := p_{ij}(W_i)$, though we will omit the dependency of W_i to improve readability.

During the task execution, we assume that each job is executed at most once, which is equivalent to saying that the execution *flow* must not contain loops. This property of our model must be ensured by the routing probabilities and implies that the task may terminate without necessarily executing all of the jobs. The fact that each job is executed at most once entails no loss of generality in practice. In fact, if a job must be executed twice, then the second execution can be modeled as a distinct job, provided that its size is independent of the original execution.

The flowchart description above can be equivalently represented by a finite Directed Acyclic Graph (DAG), where each vertex corresponds to a job, and each directed edge is weighted by a transition probability that determines the next job in the task execution sequence; see Figure 3.2 for an example. While the job sizes are only known in distribution a priori (online), the DAG topology and probabilities are fully known in advance (offline).

We denote by $G = (\mathcal{J}, \mathcal{A})$ the DAG associated with the task, where $\mathcal{J} = \{1, \ldots, K\}$ is the set of vertices (jobs), and $mathcal A = [p_{ij}]$ is the matrix of directed edges, with p_{ij} representing the routing probability from job J_i to job J_j. With a slight abuse of notation, if G is composed of only one vertex, say i, we will denote it by J_i.

Given a vertex $i \in \mathcal{J}$ (corresponding to job J_i), we denote by $\mathrm{Succ}(i)$ the set of all out-neighbors of i and by G_i the subgraph induced by i, which is again a DAG.

Remark 1 In the High-Performance Computing (HPC) literature, a DAG is typically used to represent the *workflow* of a task. In this setting, each vertex corresponds to a job, and all jobs must be executed, with the directed edges encoding precedence or synchronization constraints. In contrast, the task model considered in this paper assumes that *only a single path through the DAG is executed*. For this reason, we interpret the DAG as a *flowchart* rather than a workflow.

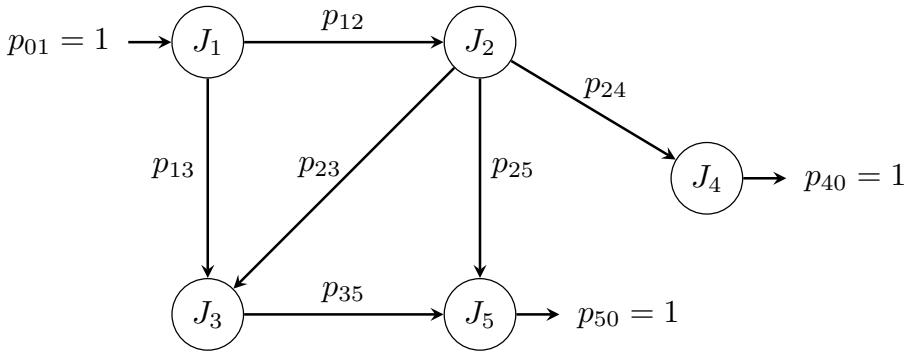

Figure 1 Example of a flowchart with four possible execution paths: $J_1 \to J_2 \to J_3 \to J_5$; $J_1 \to J_2 \to J_5$; $J_1 \to J_2 \to J_4$; $J_1 \to J_3 \to J_5$. The execution begins with job J_1 ($p_{01} = 1$). From there, job J_2 or J_3 is executed with probability p_{12} or p_{13}, respectively, and the flow continues along the DAG until no further job needs to be executed, i.e., when either J_4 or J_5 is completed.

3.3 Scheduler

The scheduler operates online and determines, at each point in time, the speed at which the processing unit should run, while ensuring that the given task is completed by the hard real-time deadline D.

We assume that speed changes are instantaneous and energy-neutral, ignoring ramp-up costs and switching delays. To construct an online speed profile, the scheduler has access to the following information:

- The overall deadline D by which the task must be completed.
- The structure of the DAG representing the task.
- The probability distributions of the job sizes.
- The set of available processing speeds.
- The power consumption function.
- The current amount of work completed for the running job, along with the completion status of each job.

The task execution terminates as soon as there is no more job to execute. At this point, the scheduler stops the processing unit. Notice that since the job sizes are unknown beforehand, the scheduler cannot predict the exact completion time. However, the scheduler learns the size of each job upon its completion and can exploit this information *online* to refine its policy in real time. Given this uncertainty, we let the scheduling decisions be based on the completed *work* rather than *time*. This is consistent with the approach followed in [5, 6]. For this reason, we let $Q(s)$ represent the dynamic power consumption *per unit of work* at speed $s \in \mathbb{R}_+$, that is

$$Q(s) := P(s)\frac{\mathrm{d}t}{\mathrm{d}w}. \tag{1}$$

Thus, $Q(s) = s^{\alpha-1}$.

3.4 Problem Statement

The objective is to determine an online speed *policy* π which, based on the current *state*, minimizes the expected energy consumption subject to the hard deadline constraint D. We begin by formulating this problem as a *continuous Markov Decision Process* (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{K}, Q)$. Here, the state space $\mathcal{S}$ is the set of tuples $(\mathbf{c}, \tau, w)$ where $\mathbf{c} \in \bigcup_{d\geq 1}(\mathcal{J} \times \mathbb{R}_+)^d$, $\tau \in \mathbb{R}_+$ and $w \in [0, \sum_{k=1}^{K} W_k^{\max}]$. The finite-dimensional real vector $\mathbf{c}$ represents the "completion list" and contains the identity and the size of the jobs that have completed (if no job is finished, then $\mathbf{c} = \emptyset$). Note that this includes the information about the total amount of work already executed. The quantity τ is the current amount of time since the beginning of the task execution process, and w is the current amount of work already executed. Note that w is not implied by $\mathbf{c}$ because it also considers the work of the job in progress. One can check that the *state* tuple $(\mathbf{c}, \tau, w)$ is indeed Markovian and corresponds to the information available to the scheduler.

Then, the action set $\mathcal{A}$ is the set of available speeds, i.e., $\mathbb{R}_+$, the transition kernel is $\mathcal{K}$, which we do not define because it will not be needed in the following, and the cost function is Q, i.e., the dynamic power consumption per unit of work.

Now, for the MDP above, we consider the total cost criterion. The mean energy consumption under policy $\pi : \mathcal{S} \to \mathbb{R}_+$ is defined as

$$\mathcal{E}_G^{\pi}(D) := \sum_{\ell \in \mathcal{L}} \mathbb{E}\left[p_\ell \int_0^{\sum_{k\in\ell} W_k} Q\big(\pi(\mathbf{c}, \tau, w)\big)\, \mathrm{d}w \right], \tag{2}$$

where G denotes the DAG underlying the structure of the given task, $\mathcal{L}$ is the set of all paths (or sequences of vertices) that the execution of the task can follow, and p_ℓ is the probability that the execution follows path ℓ (i.e., if $\ell = \{i_1, \ldots, i_L\}$, then $p_\ell = p_{0,i_1} p_{i_1,i_2} \cdots p_{i_L,0}$). Note that p_ℓ appears inside the expectation because we allow the routing probabilities to depend on the job sizes. We are interested in finding a policy π that minimizes $\mathcal{E}_G^{\pi}(D)$ while ensuring that the task is completed before D. This objective can be addressed by solving the following infinite-dimensional optimization problem:

$$\inf_{\pi} \ \mathcal{E}_G^{\pi}(D) \tag{3a}$$

$$\text{s.t.:} \ \int_0^{\sum_{k\in\ell} W_k} \frac{\mathrm{d}w}{\pi(\mathbf{c}, \tau, w)} \leq D, \quad \forall \ell \in \mathcal{L}. \tag{3b}$$

In the following, we let $\mathcal{E}_G^{*}(D) := \inf_\pi \mathcal{E}_G^{\pi}(D)$. Also, for any job k and size $w > 0$, we will frequently use the quantity

$$C_{J_k}(w) := \int_0^{w} F_k^c(u)^{\frac{1}{\alpha}}\, \mathrm{d}u, \tag{4}$$

which will play a crucial role in our approach (its meaning is clarified below).

4 Existing Results and Preliminaries

The primary objective of this section is to highlight the key challenges and subtleties hidden in the multiple-job setting. To do this, we start with a classical result for the single-job case.

4.1 Optimal Speed Profile for a Single Job

When the task is composed of a single job, J_1, the MDP formulation above simplifies drastically and the problem boils down to the setting investigated in [5]. Here, the speed policy (or profile) only depends on the amount of work already executed, and we can simply refer to it as $s_1(w) := \pi(\emptyset, \tau, w)$. Consequently, the optimization problem (3) reduces to

$$\inf_{s_1:[0,W_1^{\max}]\to\mathbb{R}_+} \int_0^{W_1^{\max}} Q(s_1(w))F_1^c(w)\mathrm{d}w \tag{5a}$$

$$\text{s.t.:} \int_0^{W_1^{\max}} \frac{\mathrm{d}w}{s_1(w)} \leq D. \tag{5b}$$

If the power function has the form $Q(s) = s^{\alpha-1}$, it is shown in [5] that the optimal solution of (5) is

$$s_1(w) = \Gamma\, F_1^c(w)^{-\frac{1}{\alpha}}, \tag{6}$$

where Γ is a (normalizing) constant such that the deadline constraint is satisfied, i.e., $\int_0^{W_1^{\max}} \frac{\mathrm{d}w}{s_1(w)} = D$. Substituting the optimal speed profile in the previous equality, Γ can be expressed by

$$\Gamma = \frac{1}{D}\int_0^{W_1^{\max}} F_1^c(w)^{\frac{1}{\alpha}}\,\mathrm{d}w = \frac{C_{J_1}(W_1^{\max})}{D}, \tag{7}$$

where $C_{J_1}(w)$ is given by (4). Thus, we can explicitly write the optimal speed profile as

$$s_1(w) = \frac{C_{J_1}(W_1^{\max})}{DF_1^c(w)^{\frac{1}{\alpha}}}. \tag{8}$$

Formula (8) shows that the optimal speed profile is increasing in the amount of completed work w, and thus also time. This is intuitive because if the deadline approaches and the job has not completed, it is natural to accelerate. Then, the optimal mean energy consumption is given by

$$\begin{aligned}\mathcal{E}^*_{J_1}(D) &= \int_0^{W_1^{\max}} Q(s_1(w))F_1^c(w)\mathrm{d}w = \frac{C_{J_1}(W_1^{\max})^{\alpha-1}}{D^{\alpha-1}}\int_0^{W_1^{\max}} \frac{F_1^c(w)}{F_1^c(w)^{\frac{\alpha-1}{\alpha}}}\mathrm{d}w \\ &= \frac{C_{J_1}(W_1^{\max})^{\alpha}}{D^{\alpha-1}}.\end{aligned} \tag{9}$$

4.2 Difficulty of the Multiple-Job Case

To illustrate the technical difficulties and subtleties of the multiple-job setting, let us suppose that the task is composed of two jobs only to be executed sequentially, first J_1 then J_2 ($p_{01} = p_{12} = p_{20} = 1$). In view of the above explicit solution for the single-job case, one may claim that the two-job case is somewhat related and easy to solve. Thus, let us introduce and discuss some *natural* policies, including one that has previously proposed in the literature. By induction, it is worth noting that the following policies extend straightforwardly to the case of $K \geq 2$ sequential jobs, without affecting the validity of the subsequent insights.

The Superjob Policy.

A first naive guess for the optimal policy consists in seeing the task as one big job, constituted as the sum of the two jobs, with probability distribution defined by the convolution of the distributions of the two jobs. Then, one can use the optimal speed profile for the single job case, i.e., (6). We call this strategy the *Superjob* policy.

The mean energy consumption induced by Superjob is

$$\mathcal{E}_G^{\text{Superjob}}(D) := \int_0^{W_1^{\max}+W_2^{\max}} Q(s_{1\star 2}(w))\,(F_1 \star F_2)^c(w)\mathrm{d}w \tag{10}$$

where $\star$ denotes the convolution operator and $s_{1\star 2}(w)$ is (6) when F_1^c is replaced by $(F_1 \star F_2)^c$

Unfortunately, the Superjob strategy is unlikely to be optimal. In fact, it fails to take into account the online information revealed upon completion of the first job while the scheduler continues executing the second one. Indeed, the distribution of the remaining work evolves at each job completion, and ignoring this dynamic leads to inefficiencies in the scheduling policy.

The Adapt Policy.

To counter argue the suboptimality argument of the Superjob policy, we consider a new policy that adapts the speed policy to the distribution of the remaining jobs upon completion of job J_1. To accomplish this, we update the speed profile every time a job is finished, so the execution of the remaining jobs can be improved. To accomplish this, we start with the speed profile of the Superjob policy and upon completion of job J_1, we execute job J_2 using the optimal speed profile for a single job, though with a modified deadline. We call this strategy the *Adapt* policy.

The mean energy consumption induced by Adapt is

$$\begin{aligned}\mathcal{E}_G^{\text{Adapt}}(D) := &\int_0^{W_1^{\max}} Q(s_{1\star 2}(w))\,F_1^c(w)\mathrm{d}w \\ &+ \int_0^{W_1^{\max}} \mathcal{E}^*_{J_2}\left(D - \int_0^w \frac{\mathrm{d}u}{s_{1\star 2}(u)}\right)\mathrm{d}F_1(w),\end{aligned} \tag{11}$$

where $s_{1\star 2}$ is as in (10). The suboptimality of Adapt is discussed below.

The d-Sweep Policy.

We now describe a policy that was used in [20]. First, the idea is to ensure that job J_1 completes by some intermediate deadline $d < D$, and to allocate the remaining time to job J_2. Here, the speed profile used for executing J_1 is chosen as the optimal profile for a single job with deadline d, while the speed profile for J_2 is selected as the optimal one for a single job with a deadline equal to the time remaining before the final deadline D. Then, the final policy is obtained by minimizing the resulting mean energy consumption over d. We call this strategy the *d-Sweep* policy.

The advantage of the d-Sweep approach is that, instead of minimizing over all possible speed profiles, the optimization reduces to the minimization of a convex one-dimensional function. This significantly simplifies the computation, as the problem becomes one-dimensional.

The mean energy consumption induced by d-Sweep is

$$\mathcal{E}_G^{d-\mathrm{Sweep}}(D) := \inf_{d\in(0,D)} \left\{ \mathcal{E}^*_{J_1}(d) + \int_0^{W_1^{\max}} \mathcal{E}^*_{J_2}\left(D - \int_0^w \frac{\mathrm{d}u}{s_1^d(u)}\right) \mathrm{d}F_1(w) \right\}. \tag{12}$$

In [20], it is claimed without proof that d-Sweep is optimal. Now, we actually provide a counterexample.

Suboptimality of Adapt and d-Sweep.

The suboptimality of Superjob has been already discussed. Somewhat counterintuitively, neither the Adapt nor the d-Sweep policy is optimal in general. We demonstrate this by providing the following counterexample.

Counterexample. Consider the case where the task is made of two jobs, $J_1 \to J_2$ where the sizes of jobs W_1 and W_2 are independent and follow an uniform distribution on $[0, 1]$. We fix the deadline to $D = 1$ and fix the power consumption per unit of work as $Q(s) = s^2$ (then $\alpha = 3$).

One can show that for the d-Sweep policy, the energy consumption is given by

$$\mathcal{E}_G^{d-\mathrm{Sweep}}(1) := \inf_{d\in(0,D)} \left\{ \frac{27}{64d^2} \left(1 + \int_0^1 \left(x^{\frac{4}{3}} + \frac{1}{d} - 1 \right)^{-2} \mathrm{d}w \right) \right\}. \tag{13}$$

We get by numerical computation that $d \approx 0.6073891$ and thus, $\mathcal{E}_G^{d-\mathrm{Sweep}}(1) \approx 2.39472$.

For the Adaptive policy, since the convolution of two independent uniform distributions on $[0, 1]$ is a triangular distribution. We numerically computed given the parameters described above, and get $\mathcal{E}_G^{\mathrm{Adapt}}(1) \approx 2.37297$.

Lastly, we provide the numerical value returned by Algorithm 1 presented later (in Section 5.3) that computes the optimal energy consumption. With a discretization step fixed at $1/N = 10^{-6}$, we get $\mathcal{E}^*_G(1) \approx 2.35347$.

5 Optimal Speed Policy for Two Jobs

In this section, we consider the optimization problem (3) under the assumption that the task is composed of two jobs only ($K = 2$). The execution starts with J_1 and then moves to J_2. It turns out that this case essentially captures the core technical difficulty of the general problem. As such, this section plays a central role, as it contains the main ideas and methods that will be extended to more complex task structures. Indeed, in the next section we will show that the general DAG case can then be solved by induction relying on the two-job case.

For conciseness, in the remainder of this section, we assume that $K = 2$ and that $p_{01} = p_{12} = p_{20} = 1$.

5.1 Characterization via the Principle of Optimality

Under the optimal policy, the key observation to exploit is that the exact size of job J_1 becomes known upon its completion. On the event $[w < W_1]$, we can write $\pi(\mathbf{c}, \tau, w)$ as a function of w only, i.e., $\pi(\mathbf{c}, \tau, w) =: s_1(w)$, as $\mathbf{c} = \emptyset$ and τ is only relevant when we compute the second job. Combining this idea with the optimality principle, we obtain the following result, which provides a convenient formulation for the mean optimal energy consumption.

Theorem 1 *Assume that $K = 2$ and that $p_{01} = p_{12} = p_{20} = 1$. Then,*

$$\mathcal{E}_G^*(D) = \inf_{s_1:[0,W_1^{\max}]\to\mathbb{R}_+^*} \int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w)\mathrm{d}w + \int_0^{W_1^{\max}} \mathcal{E}_{J_2}^*(D - \tau_1(w))\mathrm{d}F_1(w) \quad (14)$$

where

$$\tau_1(w) := \int_0^w \frac{\mathrm{d}u}{s_1(u)}, \quad (15)$$

i.e., the time of execution to compute w amount of work for the first job J_1.

Proof Under the foregoing notations, we obtain

$$\begin{aligned}\mathcal{E}_G^*(D) &= \inf_{\pi} \mathbb{E}\left[\int_0^{W_1+W_2} Q(\pi(\mathbf{c}, \tau, w))\mathrm{d}w\right] \\ &= \inf_{\pi} \mathbb{E}\left[\int_0^{W_1} Q(\pi(\mathbf{c}, \tau, w))\mathrm{d}w\right] + \mathbb{E}\left[\int_{W_1}^{W_1+W_2} Q(\pi(\mathbf{c}, \tau, w))\mathrm{d}w\right].\end{aligned}$$

The first term of the RHS term gives the mean energy consumption of the first job. Within this term, the completion list remains empty ($\mathbf{c} = \emptyset$) and the time τ brings no additional information. Thus, we can change the policy to a speed profile of the form $s_1(w) = \pi(\mathbf{c}, \tau, w)$.

The second term gives the energy consumption of the second job. This term carries the information of when the first job is finished as $\mathbf{c}$ contains the size of the first job. Given that J_1 is of size w_1, we can compute the time used to execute the first job J_1, i.e., $\tau_1(w)$ as defined in (15). Thus, under the event $[W_1 = w_1]$, the exact remaining time to compute the second job is $D - \tau_1(w_1)$. We couple this with the optimality principle, which states that under the optimal policy, the execution from the point where the first job completes onward

behaves exactly as it would if only the second job remained, with a corresponding adjusted deadline. Therefore,

$$\mathbb{E}\left[\int_{W_1}^{W_1+W_2} Q(\pi^*(\mathbf{c},\tau,w))\mathrm{d}w\right] = \int_0^{W_1^{\max}} \underbrace{\mathbb{E}\left[\int_0^{W_2} Q(\pi^*([w_1],\tau_1(w_1),w))\mathrm{d}w\right]}_{\mathcal{E}^*_{J_2}(D-\tau_1(w_1))} \mathrm{d}F_1(w_1).$$

Changing both terms and replacing $\inf_\pi$ by $\inf_{s_1}$ (as τ_1 implicitly depends deterministically on s_1), we obtain (14). □

With the new formulation in Theorem 1, to compute an optimal policy we only need to compute the speed profile for the first job s_1, as the speed profile for the second job is given by (6), i.e., the optimal speed for a single job.

5.2 Analytic Characterization

A closer analysis of the optimization in (14) brings the following result, which provides an analytic characterization for the structure of the optimal speed profile. This also highlights that its structure is highly non-trivial and conceptually distinct from the speed profiles discussed in Section 4.2.

Theorem 2 *Assume that $K = 2$ and that $p_{01} = p_{12} = p_{20} = 1$. Then, the optimal speed profile is such that*

- *the speed profile of the first job is given by*

$$s_1^*(w) = \frac{1 + \int_0^{W_1^{\max}} y(u)\mathrm{d}u}{D\ y(w)}, \tag{16}$$

where $y : [0, W_1^{\max}] \to \mathbb{R}_+$ is the solution of the second-order ordinary differential equation (with $\int y,\ y,\ y'$)

$$\frac{y'(w)}{y(w)} = \frac{1}{\alpha}\frac{\mathrm{d}F_1(w)}{F_1^c(w)}\left(C_2\left(\frac{y(w)}{1+\int_w^{W_1^{\max}} y(u)\mathrm{d}u}\right)^{\alpha} - 1\right) \tag{17}$$

and initial condition $y(W_1^{\max}) = C_2^{-1/\alpha}$ and $y'(W_1^{\max}) = 0$, where $C_2 := C_{J_2}(W_2^{\max})^{\alpha}$.
- *the speed profile of the second job is given by*

$$s_2^*(w) = \frac{C_{J_2}(W_2^{\max})}{\left(D - \int_0^{w_1} \frac{\mathrm{d}u}{s_1^*(u)}\right) F_2^c(w)^{\frac{1}{\alpha}}}, \tag{18}$$

under the event $[W_1 = w_1]$.

We stress that the optimal speed policy in Theorem 2 is online: the speed profile for the second job depends on the realized size of the first job, which becomes available

only upon completion of job J_1. At that moment, since W_1 becomes known, the processing unit can operate at speed (18), which is the formula for the optimal speed for the single job given in (8) adjusted with the correct deadline.

Proof of Theorem 2 Fix $C_2 := C_{J_2}(W_2^{\max})^\alpha$. Using the formula for $\mathcal{E}^*_{J_2}(\cdot)$ with (14), we obtain

$$\mathcal{E}^*_G(D) = \inf_{s_1:[0,W_1^{\max}]\to\mathbb{R}^*_+} \int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w)\mathrm{d}w + \int_0^{W_1^{\max}} \frac{C_2 \mathrm{d}F_1(w)}{\left(D - \int_0^w \frac{\mathrm{d}u}{s_1(u)}\right)^{\alpha-1}}. \quad (19)$$

Now we observe that (3b) cannot hold with equality as otherwise the second term in the RHS of (19) will tend to infinity. Therefore, we can drop the constraint (3b) and differentiate with respect to the function $x(w) = 1/s_1(w)$. Taking the *functional derivative* with respect to the function $x(w)$, the optimal inverse speed function $x(\cdot)$ must satisfy

$$\frac{F_1^c(w)}{x(w)^\alpha} = \int_w^{W_1^{\max}} \frac{C_2 \mathrm{d}F_1(v)}{(D - \int_0^v x(u)\mathrm{d}u)^\alpha}.$$

Now, the idea is to set $\Delta = D - \int_0^{W_1^{\max}} x(u)\mathrm{d}u$ and $y(w) = x(w)/\Delta$. In this manner, the previous equation becomes

$$\frac{F_1^c(w)}{y(w)^\alpha} = \int_w^{W_1^{\max}} \frac{C_2 \mathrm{d}F_1(v)}{(1 + \int_v^{W_1^{\max}} y(u)\mathrm{d}u)^\alpha}.$$

Differentiating both sides in w, we obtain

$$\frac{\alpha F_1^c(w) y'(w)}{y(w)^\alpha} + \frac{\mathrm{d}F_1(w)}{y(w)^\alpha} = \frac{C_2 \mathrm{d}F_1(w)}{\left(1 + \int_w^{W_1^{\max}} y(u)\mathrm{d}u\right)^\alpha}, \quad (20)$$

and reordering the terms, we finally obtain

$$\frac{y'(w)}{y(w)} = \frac{1}{\alpha}\frac{\mathrm{d}F_1(w)}{F_1^c(w)}\left(C_2\left(\frac{y(w)}{1 + \int_w^{W_1^{\max}} y(u)\mathrm{d}u}\right)^\alpha - 1\right).$$

From (20) evaluated in $W_1^{\max}$, it leads us to

$$y(W_1^{\max}) = C_2^{-\frac{1}{\alpha}}$$

and the right hand side of the ODE is equal to 0. Hence, we require that $y'(W_1^{\max}) = 0$. Finally, we recover $s_1(w)$ backwards using

$$\Delta = \frac{D}{1 + \int_0^{W_1^{\max}} y(w)\mathrm{d}w} \quad (21)$$

$$s_1(w) = \frac{1}{\Delta y(w)}. \quad (22)$$

Finally, the optimal speed for the second (and last) job follows by using the optimal speed profile for the single job (8). This concludes the proof. □

Figure 5.2 illustrates the structure of the optimal speed profile with an example where the job sizes follow a uniform distribution on $[0, 1]$ and the deadline D is fixed to 1. We observe that the speed profile increases within each job and exhibits a non-differentiable point at the transition between jobs.

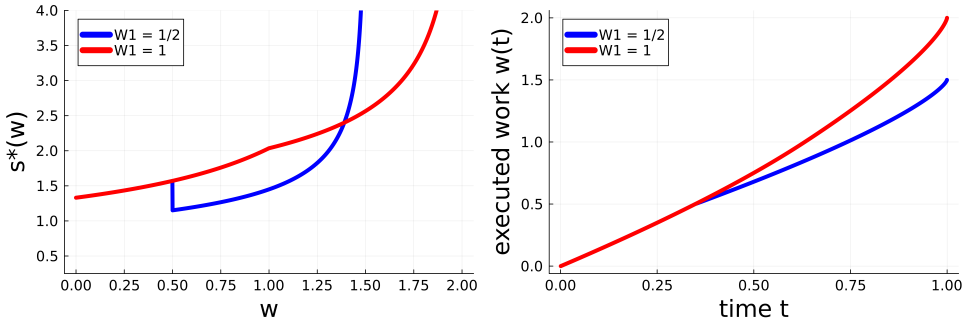

Figure 2 Optimal speed profile as a function of the executed work (left) and corresponding cumulative work executed over time (right) for two jobs with uniform distribution in $(0, 1)$ and global deadline $D = 1.$. The curves correspond to online executions where the first job W_1 has size $1/2$ (blue curve) and 1 (red curve), respectively.

5.3 Computation of the Solution via Discretization

In this section, we propose an algorithm for the computation of the optimal speed profile that relies on the analytic characterization given in Theorem 2. Instead of relying on a standard Euler discretization of the differential equation in (17), the idea is to directly discretize the continuous optimization in (19). The advantage of this approach lies in its improved efficiency, as it yields a first-order recurrence equation rather than a second-order one. To discretize the optimization problem in (19), we first partition the interval $[0, W_1^{\max}]$ into N intervals where s_1 takes values $s_1^{(1)}, \ldots, s_1^{(N)} \geq 0$, and denote by $\sigma := W_1^{\max}/N$ the length of each interval, or *discretization step*. For all $1 \leq i \leq N$, we also define $f_1^{(i)} = \mathbb{P}((i-1)\sigma \leq W_1 < i\sigma)/\sigma$ and $H_1^{(i)} := F_1^c((i-1)\sigma) = \mathbb{P}(W_1 \geq (i-1)\sigma)$, i.e., the discretized versions of the density and complementary probability distribution functions of job J_1. Then, we can rewrite the discretized optimization problem as

$$\min_{s_1^{(1)},\ldots,s_1^{(N)}>0} \quad \sigma \sum_{i=1}^{N} \left((s_1^{(i)})^{\alpha-1} H_1^{(i)} + \frac{C_2 f_1^{(i)}}{\left(D - \sigma \sum_{j=1}^{i} 1/s_1^{(j)}\right)^{\alpha-1}} \right) \tag{23a}$$

$$\text{s.t.: } \sigma \sum_{i=1}^{N} \frac{1}{s_1^{(i)}} \leq D. \tag{23b}$$

We now present Algorithm 1, which, given a parameterization of the problem, computes some speed profile.

The following result connects Algorithm 1 and Theorem 2. It states that Algorithm 1 computes the unique optimum of the optimization in (23), and that the proposed discretization approach is asymptotically correct in the limiting regime where the partition size tends to infinity.

Algorithm 1: Algorithm for the solution of (23)

Data: Deadline D, distribution functions F_1 and F_2, partition size N.

Result: Optimal speed profile $s_1^{(i)}$ for all $i = 1, \dots, N$

Compute constants $\sigma := W_1^{\max}/N$, $C_2 := C_{J_2}(W_2^{\max})^\alpha$ and $y_{(N)} := 1/\sqrt[\alpha]{C_2}$;

for $i \leftarrow N$ **to** 1 **do**

  Compute $H_1^{(i)} := \mathbb{P}(W_1 \geq (i-1)\sigma)$ and $f_1^{(i)} := \mathbb{P}((i-1)\sigma \leq W_1 < i\sigma)/\sigma$;

end

for $i \leftarrow N-1$ **to** 1 **do**

  Compute $y_{(i)}$ using

$$\frac{H_1^{(i)}}{y_{(i)}^\alpha} = \frac{H_1^{(i+1)}}{y_{(i+1)}^\alpha} + \frac{\sigma\, C_2\, f_1^{(i)}}{(1+\sigma(y_{(i+1)} + \cdots + y_{(N)}))^\alpha}. \tag{24}$$

end

Compute $\Delta = \frac{D}{1+\sigma\sum_{i=1}^N y_{(i)}}$;

for $i \leftarrow N-1$ **to** 1 **do**

  Compute

$$s_1^{(i)} = \frac{1}{\Delta y_{(i)}}. \tag{25}$$

end

Theorem 3 *The optimization problem (23) admits a unique solution, which can be computed using Algorithm 1. Moreover, the speed profile obtained by linearly interpolating the discrete solution converges uniformly, as $N \to \infty$, to the optimal speed profile given in Theorem 2.*

Proof First, we notice that an optimal solution needs to satisfy the constraint (23b) strictly, as if it would hold with equality, the second summation term in (23a) would be infinite. Analogously, in an optimal solution, all the $s_1^{(i)}$ need to be strictly positive. By complementarity slackness, this means that all the Lagrangian multipliers associated to these constraints need to be zero. Now, we make the change of variable $x_{(i)} := 1/s_1^{(i)}$. This is convenient as it is easy to see that the resulting optimization problem is strictly convex. This proves the first claim that a unique minimum exists. In addition, such minimum is obtained by imposing to zero the partial derivative in $(x_{(i)})$ of the objective function (or equivalently of the Lagrangian function because all Lagrangian multipliers are zero), for all i.

Inspired by the notations used in the proof of Theorem 2, see (21) and (22), let $\Delta = D - \sigma\sum_{i=1}^N x_{(i)}$ and let $y_{(i)} = x_{(i)}/\Delta$. Differentiating with respect to $x_{(N)}$, dividing by $\alpha - 1$ and using $y_{(i)} = x_{(i)}/\Delta$, we obtain the optimal value for $y_{(N)}$:

$$\frac{H_1^{(N)}}{y_{(N)}^\alpha} = C_2 f_1^{(N)}.$$

This implies $y_{(N)} = 1/C_2^{\frac{1}{\alpha}}$. Then, differentiating with respect to $x_{(N-1)}$, we obtain the optimal $y_{(N-1)}$:

$$\frac{H_1^{(N-1)}}{y_{(N-1)}^{\alpha}} = \sigma C_2 f_1^{(N)} + \frac{\sigma C_2 f_1^{(N-1)}}{(1+\sigma y_{(N)})^{\alpha}}.$$

Proceeding in a similar manner for all $1 \leq i < N$, we obtain (24). This inductive formula allows us to compute $y_{(N)}, y_{(N-1)}$ down to $y_{(1)}$. To compute $s_1^{(i)}$ from $y_{(i)}$, we then simply use

$$\Delta = \frac{D}{1+\sigma \sum_{i=1}^{N} y_{(i)}} \tag{26}$$

and (25). From (24), we can get back to (17). To do so, we do the following transformation,

$$\frac{H_1^{(i)} - H_1^{(i+1)}}{y_{(i)}^{\alpha}} - H_1^{(i+1)} \left(\frac{1}{y_{(i+1)}^{\alpha}} - \frac{1}{y_{(i)}^{\alpha}} \right) = \frac{\sigma C_2 f_1^{(i)}}{(1+\sigma(y_{(i+1)} + \cdots + y_{(N)}))^{\alpha}},$$

and then take the limit as the discretization step σ goes to 0, i.e., N goes to infinity, then we obtain

$$\frac{\mathrm{d}F_1(w)}{y(w)^{\alpha}} - F_1^c(w) \frac{\mathrm{d}}{\mathrm{d}w} \left(\frac{1}{y(w)^{\alpha}} \right) = \frac{C_2 \, \mathrm{d}F_1(w)}{(1+\int_w^{W_1^{\max}} y(u)\mathrm{d}u)^{\alpha}}.$$

We can then reorder the terms to recover the differential equation in (17). □

6 Optimal Speed Policy for Multiple Jobs

We now extend the approach developed in the previous section to construct an optimal speed policy for the more general case where the task follows the DAG structure described in Section 3.2. Our strategy is to first analyze the case where the DAG is just a *line*, which we handle through the principle of optimality, and then leverage this structure to construct a policy optimal for the general DAG setting. Indeed, as we will see, the DAG implicitly defines all possible execution lines (or paths), and computing the optimal speed profile reduces to taking an expectation over all these execution paths.

6.1 Sequential Jobs

Assume that all the K jobs needs to be executed sequentially, starting with J_1, i.e., $p_{01} = p_{12} = \cdots = p_{K-1,K} = p_{K0} = 1$. In this setting, the idea is to apply the same method as in Section 5 by backward induction from the last job to the first. To this end, we build on the result established in Theorem 2 and construct an optimal policy using a combination of discretization and dynamic programming techniques.

Let us outline our strategy in more detail. We begin by examining the structure of the optimal cost and decomposing it into two components, as in Theorem 1. The first term corresponds to the cost associated with the first job, while the second captures the cost incurred *from the second job onward through the remainder of the sequence.* For the first job, the completion list is empty, i.e. $\mathbf{c} = \emptyset$, the current time τ provides no additional information, and a policy is again seen as a function of w only, i.e., $s_1(w) = \pi(\mathbf{c}, \tau, w)$. For the second term, the principle of optimality ensures that the optimal policy from the point at which J_1 completes onward is equivalent to solving a

subproblem consisting of the task $(J_2, \dots, J_K)$ with a reduced deadline of $D - \tau_1(w_1)$, where w_1 denotes the realized size of job J_1.

Before stating our result, let us introduce some additional notation. Let

$$C_K := C_{J_K}(W_K^{\max})^\alpha, \tag{27}$$

and using backward induction, let us define

$$C_{k,\dots,K} := \left(1 + \int_0^{W_k^{\max}} y_k(u)\mathrm{d}u\right)^{\alpha-1} \times \left(\int_0^{W_k^{\max}} \frac{F_k^c(w)\mathrm{d}w}{y_k(w)^{\alpha-1}} + \int_0^{W_k^{\max}} \frac{C_{k+1,\dots,K}\ \mathrm{d}F_k(w)}{\left(1 + \int_w^{W_k^{\max}} y_k(u)\mathrm{d}u\right)^{\alpha-1}}\right) \tag{28}$$

for all $k < K$, where y_k is the solution of

$$\frac{y_k'(w)}{y_k(w)} = \frac{1}{\alpha}\frac{\mathrm{d}F_k(w)}{F_k^c(w)}\left(C_{k+1,\dots,K}\left(\frac{y_k(w)}{1 + \int_w^{W_k^{\max}} y_k(u)\mathrm{d}u}\right)^\alpha - 1\right)$$

and initial condition $y_k(W_k^{\max}) = C_{k+1,\dots,K}^{-1/\alpha}$ and $y_k'(W_k^{\max}) = 0$.

It is important to remark that y_k *does not depend on the job sizes*, but only on their distributions. Therefore, $y_K, y_{K-1}, \dots, y_1$ can all be computed recursively in advance and be used online to obtain the optimal speed profile.

We can now state our optimality result for sequential jobs.

Theorem 4 *Assume that* $p_{01} = p_{12} = \cdots = p_{K-1,K} = p_{K0} = 1$. *Then, the optimal energy consumption is*

$$\mathcal{E}_G^*(D) = \inf_{s_1} \int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w)\mathrm{d}w + \int_0^{W_1^{\max}} \frac{C_{2,\dots,K}}{(D - \tau_1(w))^{\alpha-1}}\mathrm{d}F_1(w) = \frac{C_{1,\dots,K}}{D^{\alpha-1}}, \tag{29}$$

where $C_{1,\dots,K}$ *is given by* (28). *In addition, for* $1 \le k < K$, *the optimal* online *speed policy for job* J_k *satisfies*

$$s_k^*(w) = \frac{1 + \int_0^{W_k^{\max}} y_k(u)\mathrm{d}u}{\left(D - \sum_{i=1}^{k-1} \tau_i(w_i)\right) y_k(w)}, \tag{30}$$

provided that $[W_1 = w_1, \dots, W_{k-1} = w_{k-1}]$, *and for the last job* J_K *satisfies*

$$s_K^*(w) = \frac{C_K F_K^c(w)^{-\frac{1}{\alpha}}}{D - \sum_{k=1}^{K-1} \tau_k(w_k)}. \tag{31}$$

Proof For $K = 2$, the statement follows by Theorem 2, so we assume $K \ge 3$.

By induction, assume that the result holds for the subgraph G_2 (induced by job J_2). Then, by the principle of optimality, and since $\pi(\emptyset, \tau, w) = s_1(w)$, we get

$$\begin{aligned}\mathcal{E}^*_G(D) &= \inf_{s_1} \int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w) \mathrm{d}w + \int_0^{W_1^{\max}} \mathcal{E}^*_{G_2}\left(D - \tau_1(w)\right) \mathrm{d}F_1(w) \\ &= \inf_{s_1} \int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w) \mathrm{d}w + \int_0^{W_1^{\max}} \frac{C_{2,\ldots,K} \mathrm{d}F_1(w)}{(D - \tau_1(w))^{\alpha-1}}\end{aligned}$$

where we used the induction hypothesis on $\mathcal{E}^*_{G_2}(D - \tau_1(w))$. Here, we observe that this is exactly the same equation as (19) by simply replacing the constant C_2 by $C_{2,\ldots,K}$. Therefore, the solution is obtained using the same differential equation as for the two job case by replacing the constant. The optimal speed profile s_k follows from (22) adjusted with the corresponding deadline, which takes into account the time used to compute the first $k-1$ jobs. □

We now address the problem of computing the optimal speed profile. Here, Algorithm 1 easily adapts to the setting of Theorem 4, as it is enough to iterate from the penultimate job to the first job in the sequence. This is done in Algorithm 2. At each step of the loop, we update the value of σ, compute $f_k^{(i)}$ and $H_k^{(i)}$ to match the corresponding job and compute $C_{k+1,\ldots,K}$ (instead of C_2) for job J_k. At the end of each loop, we compute $C_{k,\ldots,K}$ using the discretized version of (28), i.e.,

$$C_{k,\ldots,K} = \sigma \left(1 + \sigma \sum_{i=1}^{N} y_k^{(i)}\right)^{\alpha-1} \sum_{i=1}^{N} \left(\frac{H_k^{(i)}}{(y_k^{(i)})^{\alpha-1}} + \frac{C_{k+1,\ldots,K} f_k^{(i)}}{\left(1 + \sigma \sum_{j=i}^{N} y_k^{(j)}\right)^{\alpha-1}} \right). \quad (32)$$

At the end of this process, we can compute the optimal *online* speed profile by taking fixed value for the jobs size. Moreover, following the same reasoning as in the proof of Theorem 3, one can show that the discretized speed profile converges asymptotically to the optimal (continuous) speed profile as $N \to \infty$.

Note that Algorithm 2 computes *offline* the optimal policy across all possible realizations of the job size random variables. Subsequently, the scheduler applies this policy *online*, selecting the appropriate speed based on the realized job sizes as they are revealed over time.

Proceeding exactly as in the proof of Theorem 3, the speed profile curve obtained by linear interpolation of the points of the speed profile computed by Algorithm 2 is asymptotically exact as $N \to \infty$.

6.2 Task with a DAG Structure

We now consider the general case in which the task follows a DAG structure, as described in Section 3.2. Unlike the settings addressed in Sections 5 and 6.1, where the task had a total ordering of jobs, a general DAG admits only a partial ordering. This is the main issue that needs to be addressed. To this purpose, we introduce the notion of *job (or vertex) ranking*. For any vertex $i \in \mathcal{J}$ of the graph, we define the *rank* of i as the number of vertices in the longest directed path from any vertex to i. If a vertex corresponds to a starting job, i.e., it has no predecessor (or in-neighbor),

Algorithm 2: Optimal speed and energy via discretization for K sequential jobs

Data: Deadline D, distribution functions $F_1, \dots, F_K$, partition size N.
Result: Optimal speed policy and mean energy consumption.

Compute $C_K = C_{J_K}(W_K^{\max})^\alpha$;
for $k \leftarrow K-1$ **to** 1 **do**
  Compute $\sigma := W_k^{\max}/N$ and $y_k^{(N)} := 1/\sqrt[\alpha]{C_{k+1,\dots,K}}$;
  for $i \leftarrow N-1$ **to** 1 **do**
    Compute $H_k^{(i)} := \mathbb{P}(W_k \geq (i-1)\sigma)$, $f_k^{(i)} := \mathbb{P}((i-1)\sigma \leq W_k < i\sigma)/\sigma$;
  end
  for $i \leftarrow N-1$ **to** 1 **do**
    Compute $y_k^{(i)}$ using

$$\frac{H_k^{(i)}}{(y_k^{(i)})^{\alpha-1}} = \frac{H_k^{(i+1)}}{(y_k^{(i+1)})^{\alpha-1}} + \frac{C_{k+1,\dots,K} f_k^{(i)}}{\left(1 + \sigma \sum_{j=i+1}^{N} y_k^{(j)}\right)^{\alpha-1}}$$

    i.e., (24) written for job J_k instead of job J_1;
  end
  Compute $C_{k,\dots,K}$ using (32);
end

then it has rank one. We denote the rank of i by $R(i)$ and we let $R_{\max} := \max_i R(i)$ be the maximal rank. This ranking induces a partial order over the jobs and enables the computation of the mean energy consumption via backward induction on the job ranks. For example, for the graph in Figure 3.2, $R(1) = 1$, $R(2) = 2$, $R(3) = 3$ and $R(4) = R(5) = 4$. For simplicity, we assume that the DAG has only one starting job, J_1, i.e., $p_{01} = 1$. Thus, $G = G_1$.

Job rankings can be efficiently computed in $O(A)$ time, where A denotes the total number of edges in the DAG G. This can be achieved using Algorithm 3, which traverses each edge exactly once during the execution of the `while` loop. Indeed, an edge is only visited when its source vertex is removed from the graph. Moreover, since G is acyclic and acyclicity is a hereditary property, every subgraph of G necessarily

contains at least one vertex with no incoming edges. Hence, we have $|\mathcal{J}_{r+1}| < |\mathcal{J}_r|$.

Algorithm 3: Ranking algorithm for a DAG

Data: DAG $G = (\mathcal{J}, \mathcal{A})$
Result: Partial ordering rank function $R : \mathcal{J} \to \mathbb{N}$
$(\mathcal{J}_1, \mathcal{A}_1) \leftarrow (\mathcal{J}, \mathcal{A})$, and $\mathcal{R}_0 \leftarrow \emptyset$;
$r \leftarrow 1$;
while $(\mathcal{J}_r, \mathcal{A}_r) \neq \emptyset$ **do**
  Give rank r to all vertices that have no predecessors in graph $(\mathcal{J}_r, \mathcal{A}_r)$;
  $\mathcal{R}_r \leftarrow \{i \in \mathcal{J}_r \mid i \text{ has rank } r\}$;
  $\mathcal{J}_{r+1} \leftarrow \mathcal{J}_r \setminus \mathcal{R}_r$ and $\mathcal{A}_{r+1} \leftarrow \mathcal{A}_r \setminus \{uv \in \mathcal{A}_r \mid u \in \mathcal{R}_r \text{ or } v \in \mathcal{R}_r\}$;
  $r \leftarrow r + 1$.
end
Define for all $i \in \mathcal{J}$, $R(i) = r$ if $i \in \mathcal{R}_r$.

We recall that $\mathrm{Succ}(i)$ denotes the set of all out-neighbors of vertex i, and observe that the rank of any successor of a vertex is strictly greater than the rank of the vertex itself. Consequently, when the subgraph G_i consists of a single vertex, we recover the single-job case previously analyzed in Section 4.1.

For the subgraph G_i induced by vertex i, we first rewrite (2) recursively as

$$\mathcal{E}^{\pi}_{G_i}(D) = \mathbb{E}\left[\int_0^{W_i} Q(\pi(\mathbf{c}, \tau, w))\mathrm{d}w + \sum_{j \in \mathrm{Succ}(i)} p_{ij}(W_i)\, \mathcal{E}^{\pi}_{G_j}(D - \tau_i(W_i))\right]. \tag{33}$$

Then, for every $i \in \mathcal{J}$, we let (similarly to (28))

$$C_{G_i} := \left(1 + \int_0^{W_i^{\max}} y_i(u)\mathrm{d}u\right)^{\alpha-1} \times \left(\int_0^{W_i^{\max}} \frac{F_i^c(w)\mathrm{d}w}{y_i(w)^{\alpha-1}} + \int_0^{W_i^{\max}} \frac{\sum_{j \in \mathrm{Succ}(i)} p_{ij}(w) C_{G_j}\ \mathrm{d}F_i(w)}{\left(1 + \int_w^{W_i^{\max}} y_i(u)\mathrm{d}u\right)^{\alpha-1}}\right), \tag{34}$$

where y_i is the solution of

$$\frac{y_i'(w)}{y_i(w)} = \frac{1}{\alpha}\frac{\mathrm{d}F_i(w)}{F_i^c(w)}\left(\sum_{j \in \mathrm{Succ}(i)} p_{ij}(w) C_{G_j}\left(\frac{y_i(w)}{1 + \int_w^{W_i^{\max}} y_i(u)\mathrm{d}u}\right)^{\alpha} - 1\right)$$

and initial condition $y_i(W_i^{\max}) = \left(\sum_{j \in \mathrm{Succ}(i)} p_{ij}(W_i^{\max}) C_{G_j}\right)^{-1/\alpha}$ and $y_i'(W_i^{\max}) = 0$.

Again, C_{G_i} is a constant that can be computed by backward induction on the ranks. Now, applying (33) to the entire DAG and using the optimality principle as done in Section 5, we obtain the following result.

Theorem 5 *If J_1 is the unique starting job of the task ($p_{01} = 1$), then the optimal mean energy consumption satisfies*

$$\begin{aligned}\mathcal{E}^*_{G_1}(D) &= \frac{C_{G_1}}{D^{\alpha-1}} \\ &= \inf_{s_1}\left(\int_0^{W_1^{\max}} s_1(w)^{\alpha-1} F_1^c(w)\mathrm{d}w + \int_0^{W_1^{\max}} \frac{\sum_{j\in\mathrm{Succ}(1)} p_{1j}(w_1)\, C^{\alpha}_{G_j}}{(D-\tau_1(w_1))^{\alpha-1}}\mathrm{d}F_1(w_1)\right). \end{aligned} \tag{35}$$

In addition, the online *optimal speed profile for job J_i is*

$$s_i^*(w) = \frac{1+\int_0^{W_i^{\max}} y_i(u)\mathrm{d}u}{\left(D-\sum_{\ell=1}^{k}\tau_{i_\ell}(w_{i_\ell})\right) y_i(w)}. \tag{36}$$

provided that $[W_{i_1} = w_{i_1}, \ldots, W_{i_k} = w_{i_k}]$ and $J_{i_1}, \ldots, J_{i_k}$ are the jobs executed before J_i.

Proof The proof is identical to that of Theorem 4, with the exception of the constants C_{G_i}, which vary depending on the number of successors in the current subgraph under consideration. □

In view of Theorem 5, the optimal online speed profile can be computed via forward induction. It is important to note that the transition probabilities are *online* quantities, as they depend on the realized size of the most recently completed job (recall that $p_{ij} = p_{ij}(W_i)$).

6.3 Efficient Computation

Algorithm 4 provides a general algorithm to compute efficiently the optimal mean energy consumption and the optimal speed policy for task with a general DAG structure. It is a direct generalization of Algorithm 1. As such, following the same reasoning as in the proof of Theorem 3, the discretized speed profile is asymptotically correct as $N \to \infty$.

Algorithm 4: Computation of an optimal speed policy and mean energy consumption in the DAG setting.

Data: Deadline D, DAG $G = (\mathcal{J}, \mathcal{A})$, distribution functions $F_1, \dots, F_K$, partition size N.
Result: Optimal policy and mean energy consumption.

```
1  Rank the vertices of the graph using Algorithm 3;
2  for r ← R_max to 1 do
3      forall i ∈ J, R(i) = r do
4          if J_i is a final job then
5              C_{G_i} = C_{J_i}(W_i^max)^α
6          else
7              σ := W_i^max / N;
8              forall j ← 1 to N do
9                  H_k^(j) := P(W_i ≥ (j−1)σ) and
                   f_i^(j) := P((j−1)σ ≤ W_i < jσ)/σ;
10             end
11             y_i^(N) := Σ_{ℓ∈Succ(i)} p_iℓ((N−1)σ) C_{G_ℓ};
12             for j ← N−1 to 1 do
13                 Compute y_i^(j) using Theorem 5 (p_iℓ^(j) = p_iℓ((j−1)σ));
14             end
15             Compute C_{G_i} using (34);
16         end
17     end
18 end
19 Compute E*_{G_1}(D) using (35);
```

Finally, we remark that Algorithm 4 is computationally efficient, even for large tasks. In fact, its computational complexity is linear in the number of edges of the underlying DAG and in the number of discretization steps (N). This is stated in the next proposition.

Proposition 6 *Algorithm 4 runs in $O(AN)$ steps, where A is the number of edges in the DAG G and N is the discretization size.*

Proof As previously established, Algorithm 3 (line 1) runs in time $O(A)$. The complexity of Algorithm 4 is due to its innermost loop (line 13), that involves a sum over all successors of i. Summing over all discretization steps and all jobs gives a complexity in $O(AN)$. Actually, a closer look at the algorithm shows that the complexity can be further detailed as $(C_1 K_{(I)} + C_2 A)N$, where $K_{(I)}$ is the number of non-terminal jobs and C_1, C_2 are constants. □

To support Proposition 6, we report numerical results obtained by running Algorithm 4 on the DAG in Figure 3.2 (with resp. its first 2,3,4 and 5 jobs). We record the minimum observed execution time over 10,000 runs, which serves as the closest proxy for the true runtime since it is least affected by transient CPU noise; this is standard

N	2 Jobs	3 Jobs	4 Jobs	5 Jobs
100	$203.106\,\mu s$	$418.377\,\mu s$	$465.634\,\mu s$	$698.510\,\mu s$
1000	$2.504\,ms$	$5.189\,ms$	$5.329\,ms$	$7.991\,ms$
10000	$73.500\,ms$	$151.993\,ms$	$154.520\,ms$	$234.514\,ms$

Table 1 Minimum execution time over 10 000 runs of Algorithm 4 for the DAG in Figure 3.2. Transition probabilities are fixed as $p_{12} = 0.6$, $p_{13} = 0.4$, $p_{23} = 0.2$, $p_{24} = 0.5$, and $p_{25} = 0.3$ and all jobs have a uniform distribution in $[0, 1]$ The column "k Jobs" ($k = 2, 3, 4, 5$) corresponds to the DAG in Figure 3.2 restricted to its first k jobs.

practice when estimating execution time from repeated measurements. The algorithm was implemented in Julia and was run over a desktop computer. The corresponding results are reported in Table 1. They are consistent with Proposition 6: going from 2 to 3 jobs adds two more egdes in the DAG, from 3 to 4 jobs only one more egde (hence a lesser complexity gap than from 2 to 3) and going from 4 to 5 jobs adds two more edges again. In fact the gaps between the columns is even better explained by the detailled complexity of the algorithm given in the proof of Proposition 6. Using $N = 1000$ usually provides a good enough precision and first quite fast, making our algorithm well suited for computing the optimal speed over large scale tasks.

7 Conclusion and Future Work

In this paper, we studied the problem of minimizing energy consumption for the execution of a task composed of multiple jobs on a standard model of a processor and under a hard real-time deadline constraint. Starting from the classical result in [5] for a single stochastic job, we extended the framework to tasks consisting of multiple jobs executed sequentially, and further to general flowchart-based structures modeled as DAGs. Our analysis revealed the structure of the optimal speed policy in each case and showed that, although a closed-form solution is not always available, the optimal policy can be computed efficiently through dynamic programming and discretization techniques. In particular, we developed scalable algorithms with provable guarantees and linear complexity in the number of jobs and discretization steps, making them suitable for practical implementations.

This work opens several promising directions for future research:

- *Generalized power functions.* We plan to investigate how the structure of the optimal policy changes when the power consumption function is generalized beyond the specific forms considered here, particularly to arbitrary convex and increasing functions.
- *Discrete and finite speed sets.* Our current model assumes a processor with a continuous speed range and leaves unsolved the setting where the set of available speeds is discrete. Understanding how to design an optimal speed profile in this setting has been partially addressed in [6], though only for the case where the task consists of a single job.
- *Preemptive scheduling.* An important extension is the study of preemptive policies, where the scheduler may interrupt the execution of one job to start or resume

another. This adds significant flexibility and realism, especially when jobs vary greatly in size or urgency. The design of optimal preemptive policies – possibly with switching costs – remains largely unexplored in this context.

- *Learning job distributions.* In many practical settings, the job size distributions are not known a priori but can be learned over time from past observations. This naturally leads to online learning frameworks where the scheduler dynamically estimates distributional properties and adapts its policy accordingly. Future work will explore such settings, leveraging tools from bandit theory and reinforcement learning to combine statistical estimation with real-time scheduling.

Acknowledgements. This work has been funded by the projects DynamicEnergy EMERGENCE 2025 "Reinforcement Learning for Dynamic Minimization of Energy Consumption in Real Time Embedded Systems" and FOUNDRY-ANR-23-PEIA-0003.

Declarations.

- The authors declare no conflict of interest.
- This article contains no studies performed by authors with human participants or animals.

References

[1] Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Korhonen, A., Traum, D., Màrquez, L. (eds.) Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650. Association for Computational Linguistics, Florence, Italy (2019). https://doi.org/10.18653/v1/P19-1355 . https://aclanthology.org/P19-1355/

[2] Khaledian, N., Khamforoosh, K., Akraminejad, R., Abualigah, L., Javaheri, D.: An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment. Computing **106**(1), 109–137 (2023) https://doi.org/10.1007/s00607-023-01215-4

[3] Sedghani, H., Filippini, F., Ardagna, D.: Space4ai-d: A design-time tool for ai applications resource selection in computing continua. IEEE Transactions on Services Computing **17**(6), 4324–4339 (2024) https://doi.org/10.1109/TSC.2024.3479935

[4] Hussain, M., Luo, M.-X., Hussain, A., Javed, M.H., Abbas, Z., Wei, L.-F.: Deadline-constrained cost-aware workflow scheduling in hybrid cloud. Simulation Modelling Practice and Theory **129**, 102819 (2023) https://doi.org/10.1016/j.simpat.2023.102819

[5] Lorch, J.R., Smith, A.J.: Improving dynamic voltage scaling algorithms with PACE. In: Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS'01, pp. 50–61. ACM, Cambridge (MA), USA (2001)

[6] Anselmi, J., Gaujal, B.: Energy optimal activation of processors for the execution of a single task with unknown size. In: 2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 65–72 (2022). https://doi.org/10.1109/MASCOTS56607.2022.00017

[7] Jones, N.: How to stop data centres from gobbling up the world's electricity. Nature **561**(7722), 163–166 (2018) https://doi.org/10.1038/d41586-018-06610-y

[8] Ericsson: Mobility report: Sustainability and ict – 2020 outcome. Technical report, Ericsson (2020). "https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/ict-sustainability"

[9] Hu, B., Yang, X., Zhao, M.: Online energy-efficient scheduling of dag tasks on heterogeneous embedded platforms. Journal of Systems Architecture **140**, 102894 (2023) https://doi.org/10.1016/j.sysarc.2023.102894

[10] Shioura, A., Shakhlevich, N., Strusevich, V.: Machine speed scaling by adapting methods for convex optimization with submodular constraints. INFORMS Journal on Computing **29**(4), 724–736 (2017) https://doi.org/10.1287/ijoc.2017.0788

[11] Gupta, A., Krishnaswamy, R., Pruhs, K.: Scalably scheduling power-heterogeneous processors. In: Abramsky, S., Gavoille, C., Kirchner, C., Heide, F., Spirakis, P.G. (eds.) Automata, Languages and Programming, pp. 312–323. Springer, Berlin, Heidelberg (2010)

[12] Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of IEEE Annual Foundations of Computer Science, pp. 374–382 (1995)

[13] Li, M., Yao, F.F., Yuan, H.: An $O(n^2)$ algorithm for computing optimal continuous voltage schedules. In: TAMC'17. LNCS, vol. 10185, pp. 389–400. Bern, Switzerland (2017)

[14] Gaujal, B., Girault, A., Plassart, S.: A Pseudo-Linear Time Algorithm for the Optimal Discrete Speed Minimizing Energy Consumption. Discrete Event Dynamic Systems **31**, 163–184 (2021) https://doi.org/10.1007/s10626-020-00327-9

[15] Anselmi, J., Gaujal, B.: Energy-optimal scheduling with variable processing speed: The role of task size variability. Operations Research Letters **64**, 107379 (2026) https://doi.org/10.1016/j.orl.2025.107379

[16] Gaujal, B., Girault, A., Plassart, S.: Dynamic speed scaling minimizing expected energy consumption for real-time tasks. J. of Scheduling **23**(5), 555–574 (2020)

[17] Anselmi, J., Gaujal, B., Rebuffi, L.S.: Optimal Speed Profile of a DVFS Processor

under Soft Deadlines. Performance Evaluation **152** (2021) https://doi.org/10.1016/j.peva.2021.102245

[18] Filippini, F., Anselmi, J., Ardagna, D., Gaujal, B.: A stochastic approach for scheduling AI training jobs in gpu-based systems. IEEE Trans. Cloud Comput. **12**(1), 53–69 (2024) https://doi.org/10.1109/TCC.2023.3336540

[19] Xu, R., Mossé, D., Melhem, R.: Minimizing expected energy in real-time embedded systems. In: Proceedings of the 5th ACM International Conference on Embedded Software. EMSOFT '05, pp. 251–254. Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1086228.1086274 . https://doi.org/10.1145/1086228.1086274

[20] Lu, Z., Zhang, Y., Stan, M., Lach, K., Skadron, K.: Procrastinating voltage scheduling with discrete frequency sets. 2008 Design, Automation and Test in Europe **1**, 98 (2006) https://doi.org/10.1109/DATE.2006.243828

[21] Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM Journal on Computing **39**(4), 1294–1308 (2009) https://doi.org/10.1137/070706793