Efficient Computation of Optimal Energy and Fractional Weighted Flow Trade-off Schedules

Antonios Antoniadis[*1], Neal Barcelo[†2], Mario Consuegra[‡3], Peter Kling[§4], Michael Nugent[5], Kirk Pruhs[¶6], and Michele Scquizzato[‖7]

1,2,5,6,7 University of Pittsburgh, Pittsburgh, USA
3 Florida International University, Miami, USA
4 University of Paderborn, Paderborn, Germany

Abstract

We give a polynomial time algorithm to compute an optimal energy and fractional weighted flow trade-off schedule for a speed-scalable processor with discrete speeds. Our algorithm uses a geometric approach that is based on structural properties obtained from a primal-dual formulation of the problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases scheduling, flow time, energy efficiency, speed scaling, primal-dual

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.63

1 Introduction

It seems to be a universal law of technology in general, and information technology in particular, that higher performance comes at the cost of energy efficiency. Thus a common theme of green computing research is how to manage information technologies so as to obtain the proper balance between these conflicting goals of performance and energy efficiency. Here the technology we consider is a speed-scalable processor, as manufactured by the likes of Intel and AMD, that can operate in different modes, where each mode has a different speed and power consumption, and the higher speed modes are less energy-efficient in that they consume more energy per unit of computation. The management problem that we consider is how to schedule jobs on such a speed-scalable processor in order to obtain an optimal trade-off between a natural performance measure (fractional weighted flow) and the energy used. Our main result is a polynomial time algorithm to compute such an optimal trade-off schedule.

We want to informally elaborate on the statement of our main result. Fully formal definitions can be found in Section 3. We need to explain how we model the processors, the jobs, a schedule, our performance measure, and the energy-performance trade-off:

* Supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
† This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1247842.
‡ Supported by NSF Graduate Research Fellowship DGE-1038321.
§ Supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the Graduate School on Applied Network Science (GSANS).
¶ Supported in part by NSF grants CCF-1115575, CNS-1253218 and an IBM Faculty Award.
‖ Supported in part by a fellowship of "Fondazione Ing. Aldo Gini", University of Padova, Italy.

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 63–74

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Speed-Scalable Processor: We assume that the processor can operate in any of a discrete set of modes, each with a specified speed and power consumption.

The Jobs: Each job has a release time when the job arrives in the system, a volume of work (think of a unit of work as being an infinitesimally small instruction to be executed), and a total importance or weight. The ratio of the weight to the volume of work specifies the density of the job, which is the importance per unit of work of that job.

A Schedule: A schedule specifies, for each real time, the job that is being processed and the mode of the processor.

Our Performance Measure of Fractional Weighted Flow: The fractional weighted flow of a schedule is the total over all units of work (instructions) of how much time that work had to wait from its release time until that work was executed on the processor, times the weight (aggregate importance) of that unit of work. So work with higher weight is considered to be more important. Presumably the weights are specified by higher-level applications that have knowledge of the relative importance of various jobs.

Optimal Trade-off Schedule: An optimal trade-off schedule minimizes the fractional weighted flow plus the energy used by the processor (energy is just power integrated over time). To gain intuition, assume that at time zero a volume p of work of weight w is released. Intuitively/Heuristically one might think that the processor should operate in the mode i that minimizes $w\frac{p}{2s_i} + P_i\frac{p}{s_i}$, where s_i and P_i are the speed and power of mode i respectively, until all the work is completed; In this schedule the time to finish all the work is $\frac{p}{s_i}$, the fractional weighted flow is $w\frac{p}{2s_i}$, and the total energy usage is $P_i\frac{p}{s_i}$. So the larger the weight w, the faster the mode that the processor will operate in. Thus intuitively the application-provided weights inform the system scheduler as to which mode to operate in so as to obtain the best trade-off between energy and performance. (The true optimal trade-off schedule for the above instance is more complicated as the speed will decrease as the work is completed.)

In Section 2 we explain the relationship of our result to related results in the literature. Unfortunately both the design and analysis of our algorithm are complicated, so in Section 4 we give an overview of the main conceptual ideas before launching into details in the subsequent sections. In Section 5 we present the obvious linear programming formulation of the problem, and discuss our interpretation of information that can be gained about optimal schedules from both the primal and dual linear programs. In Section 6 we use this information to develop our algorithm. Finally in Section 7 we analyze the running time of our algorithm. Due to space limitations, many of the details are left for the full version of the paper.

2 Related Results

To the best of our knowledge there are three papers in the algorithmic literature that study computing optimal energy trade-off schedules. All of these papers assume that the processor can run at any non-negative real speed, and that the power used by the processor is some nice function of the speed, most commonly the power is equal to the speed raised to some constant α. Essentially both [2, 13] give polynomial time algorithms for the special case of our problem where the densities of all units of work are the same. The algorithm in [13] is a homotopic optimization algorithm that intuitively traces out all schedules that are Pareto-optimal with respect to energy and fractional flow, one of which must obviously be the optimal energy trade-off schedule. The algorithm in [2] is a dynamic programming algorithm. [2] also deserves credit for introducing the notion of trade-off schedules. [7] gave

a polynomial-time algorithm for recognizing an optimal schedule. [7] also showed that the optimal schedule evolves continuously as a function of the importance of energy, implying that a continuous homotopic algorithm is, at least in principle, possible. However, [7] was not able to provide any bound, even exponential, on the time of this algorithm, nor was [7] able to provide any way to discretize this algorithm.

To reemphasize, the prior literature [2, 13, 7] on our problem assumes that the set of allowable speeds is continuous. Our setting of discrete speeds both more closely models the current technology, and seems to be algorithmically more challenging. In [7] the recognition of an optimal trade-off schedule in the continuous setting is essentially a direct consequence of the KKT conditions of the natural convex program, as it is observed that there is essentially only one degree of freedom for each job in any plausibly optimal schedule, and this degree of freedom can be recovered from the candidate schedule by looking at the speed that the job is run at any time that the job is run. In the discrete setting, we shall see that there is again essentially only one degree of freedom for each job, but unfortunately one cannot easily recover the value of this degree of freedom by examining the candidate schedule. Thus we do not know of any simple way to even recognize an optimal trade-off schedule in the discrete setting.

One might also reasonably consider the performance measure of the aggregate weighted flow over jobs (instead of work), where the flow of a job is the amount of time between when the job is released and when the last bit of work of that job is finished. In the context that the jobs are flight queries to a travel site, aggregating over the delay of jobs is probably more appropriate in the case of Orbitz, as Orbitz does not present the querier with any information until all the possible flights are available, while aggregating over the delay of work may be more appropriate in the case of Kayak, as Kayak presents the querier with flight options as they are found. Also, often the aggregate flow of work is used as a surrogate measure for the aggregate flow of jobs as it tends to be more mathematically tractable. In particular, for the trade-off problem that we consider here, the problem is NP-hard if we were to consider the performance measure of the aggregate weighted flow of jobs, instead of the aggregate weighted flow of work. The hardness follows immediately from the well known fact that minimizing the weighted flow time of jobs on a unit speed processor is NP-hard [10], or from the fact that minimizing total weighted flow, without release times, subject to an energy budget is NP-hard [12].

There is a fair number of papers that study approximately computing optimal trade-off schedules, both offline and online. [12] also gives PTAS's for minimizing total flow without release times subject to an energy budget in both the continuous and discrete speed settings. [2, 6, 11, 4, 3, 5, 8, 9] consider online algorithms for optimal total flow and energy, [4, 5] consider online algorithms for fractional flow and energy. For a survey on energy-efficient algorithms, see [1].

3 Model & Preliminaries

We consider the problem of scheduling a set $\mathcal{J} \coloneqq \{1, 2, \ldots, n\}$ of n jobs on a single processor featuring k different speeds $0 < s_1 < s_2 < \ldots < s_k$. The power consumption of the processor while running at speed s_i is $P_i \geq 0$. We use $\mathcal{S} \coloneqq \{s_1, \ldots, s_k\}$ to denote the set of speeds and $\mathcal{P} \coloneqq \{P_1, \ldots, P_k\}$ to denote the set of powers. While running at speed s_i, the processor performs s_i units of work per time unit and consumes energy at a rate of P_i.

Each job $j \in \mathcal{J}$ has a release time r_j, a processing volume (or work) p_j, and a weight w_j. Moreover, we denote the value $d_j \coloneqq \frac{w_j}{p_j}$ as the density of job j. All densities are distinct;

details about this assumption are left for the full version. For each time t, a schedule S must decide which job to process at what speed. We allow preemption, that is, a job may be suspended at any point in time and resumed later on. We model a schedule S by a speed function $V\colon \mathbb{R}_{\geq 0} \to \mathcal{S}$ and a scheduling policy $J\colon \mathbb{R}_{\geq 0} \to \mathcal{J}$. Here, $V(t)$ denotes the speed at time t, and $J(t)$ the job that is scheduled at time t. Jobs can be processed only after they have been released. For job j let $I_j = J^{-1}(j) \cap [r_j, \infty)$ be the set of times during which it is processed. A feasible schedule must finish the work of all jobs. That is, the inequality $\int_{I_j} S(t)\,\mathrm{d}t \geq p_j$ must hold for all jobs j.

We measure the quality of a given schedule S by means of its energy consumption and its fractional flow. The speed function V induces a power function $P\colon \mathbb{R}_{\geq 0} \to \mathcal{P}$, such that $P(t)$ is the power consumed at time t. The energy consumption of schedule S is $E(S) \coloneqq \int_0^\infty P(t)\,\mathrm{d}t$. The flow time (also called response time) of a job j is the difference between its completion time and release time. If F_j denotes the flow time of job j, the weighted flow of schedule S is $\sum_{j\in J} w_j F_j$. However, we are interested in the fractional flow, which takes into account that different parts of a job j finish at different times. More formally, if $p_j(t)$ denotes the work of job j that is processed at time t (i.e., $p_j(t) = V(t)$ if $J(t) = j$, and $p_j(t) = 0$ otherwise), the fractional flow time of job j is $\widetilde{F}_j \coloneqq \int_{r_j}^\infty (t - r_j)\frac{p_j(t)}{p_j}\,\mathrm{d}t$. The fractional weighted flow of schedule S is $\widetilde{F}(S) \coloneqq \sum_{j\in\mathcal{J}} w_j \widetilde{F}_j$. The objective function is $E(S) + \widetilde{F}(S)$. Our goal is to find a feasible schedule that minimizes this objective.

We define $s_0 \coloneqq 0$, $P_0 \coloneqq 0$, $s_{k+1} \coloneqq s_k$, and $P_{k+1} \coloneqq \infty$ to simplify notation. Note that, without loss of generality, we can assume $\frac{P_i - P_{i-1}}{s_i - s_{i-1}} < \frac{P_{i+1} - P_i}{s_{i+1} - s_i}$; Otherwise, any schedule using s_i could be improved by linearly interpolating the speeds s_{i-1} and s_{i+1}.

4 Overview

In this section we give an overview of our algorithm design and analysis. We start by considering a natural linear programming formulation of the problem. We then consider the dual linear program. Using complementary slackness we find necessary and sufficient conditions for a candidate schedule to be optimal. Reminiscent of the approach used in the case of continuous speeds in [7], we then interpret these conditions in the following geometric manner. Each job j is associated with a linear function $D_j^{\alpha_j}(t)$, which we call *dual line*. This dual line has a slope of $-d_j$ and passes through point (r_j, α_j), for some $\alpha_j > 0$. Here t is time, α_j is the dual variable associated with the primal constraint that all the work from job j must be completed, r_j is the release time of job j, and d_j is the density of job j. Given such an α_j for each job j, one can obtain an associated schedule as follows: At every time t, the job j being processed is the one whose dual line is the highest at that time, and the speed of the processor depends solely on the height of this dual line at that time.

The left picture in Figure 1 shows the dual lines for four different jobs on a processor with three modes. The horizontal axis is time. The two horizontal dashed lines labeled by C_2 and C_3 represent the heights where the speed will transition between the lowest speed mode and the middle speed mode, and the middle speed mode and the highest speed mode, respectively (these lines only depend on the speeds and powers of the modes and not on the jobs). The right picture in Figure 1 shows the associated schedule.

By complementary slackness, a schedule corresponding to a collection of α_j's is optimal if and only if it processes exactly p_j units of work for each job j. Thus we can reduce finding an optimal schedule to finding values for these dual variables with this property.

Our algorithm is a primal-dual algorithm that raises the dual α_j variables in an organized way. We iteratively consider the jobs by decreasing density. In iteration i, we construct the

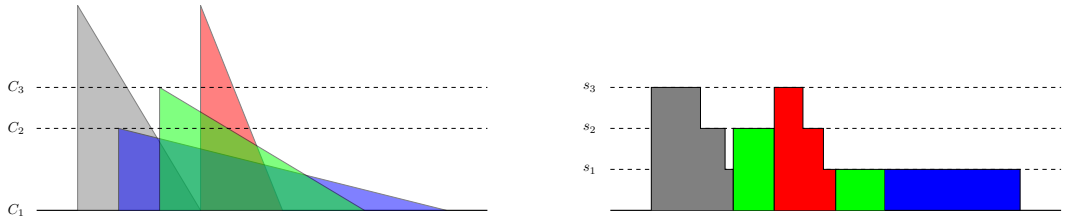

Figure 1 The dual lines for a 4-job instance, and the associated schedule.

optimal schedule S_i for the i most dense jobs from the optimal schedule S_{i-1} for the $i-1$ most dense jobs. We raise the new dual variable α_i from 0 until the associated schedule processes p_i units of work from job i. At some point raising the dual variable α_i may cause the dual line for i to "affect" the dual line for a previous job j in the sense that α_j must be raised as α_i is raised in order to maintain the invariant that the right amount of work is processed on job j. Intuitively one might think of "affection" as meaning that the dual lines intersect (this is not strictly correct, but might be a useful initial geometric interpretation to gain intuition). More generally this affection relation can be transitive in the sense that raising the dual variable α_j may in turn affect another job, etc.

The algorithm maintains an affection tree rooted at i that describes the affection relationship between jobs, and maintains for each edge in the tree a variable describing the relative rates that the two incident jobs must be raised in order to maintain the invariant that the proper amount of work is processed for each job. Thus this tree describes the rates that the dual variables of old jobs must be raised as the new dual variable α_i is raised at a unit rate.

In order to discretize the raising of the dual lines, we define four types of events that cause a modification to the affection tree:

- a pair of jobs either begin or cease to affect each other,
- a job either starts using a new mode or stops using some mode,
- the rightmost point on a dual line crosses the release time of another job, or
- enough work is processed on the new job i.

During an iteration, the algorithm repeatedly computes when the next such event will occur, raises the dual lines until this event, and then computes the new affection tree. Iteration i completes when job i has processed enough work. Its correctness follows from the facts that (i) the affection graph is a tree, (ii) this affection tree is correctly computed, (iii) the four aforementioned events are exactly the ones that change the affection tree, and (iv) the next such event is correctly computed by the algorithm. We bound the running time by bounding the number of events that can occur, the time required to calculate the next event of each type, and the time required to recompute the affection tree after each event.

5 Structural Properties via Primal-Dual Formulation

In the following, we give an integer linear programming (ILP) description of our problem. To this end, let us assume that time is divided into discrete time slots such that, in each time slot, the processor runs at constant speed and processes at most one job. Note that these time slots may be arbitrarily small, yielding an ILP with many variables and, thus, rendering a direct solution approach less attractive. However, we are actually not interested in solving this ILP directly. Instead, we merely strive to use it and its dual in order to obtain some simple structural properties of an optimal schedule.

$$\begin{aligned}\min \quad & \sum_{j\in\mathcal{J}}\sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\big(P_i + s_i d_j(t - r_j + 1/2)\big) \\ \text{s.t.} \quad & \sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\cdot s_i \geq p_j && \forall j \\ & \sum_{j\in\mathcal{J}}\sum_{i=1}^{k} x_{jti} \leq 1 && \forall t \\ & x_{jti} \in \{0,1\} && \forall j,t,i\end{aligned}$$

(a) ILP formulation of our scheduling problem.

$$\begin{aligned}\max \quad & \sum_{j\in\mathcal{J}} p_j\alpha_j - \sum_{t=1}^{T}\beta_t \\ \text{s.t.} \quad & \beta_t \geq \alpha_j s_i - P_i \\ & \qquad -s_i d_j(t - r_j + 1/2) \\ & \qquad \forall j,t,i : t \geq r_j \\ & \alpha_j \geq 0 \quad \forall j \\ & \beta_t \geq 0 \quad \forall t\end{aligned}$$

(b) Dual program of the ILP's relaxation.

Figure 2

ILP & Dual Program. Let the indicator variable x_{jti} denote whether job j is processed in slot t at speed s_i. Moreover, let T be some upper bound on the total number of time slots. This allows us to model our scheduling problem via the ILP given in Figure 2a. The first set of constraints ensures that all jobs are completed, while the second set of constraints ensures that the processor runs at constant speed and processes at most one job in each time slot.

In order to use properties of duality, we consider the relaxation of the above ILP. It can easily be shown that any optimal schedule will always use highest density first as its scheduling policy, and therefore there is no advantage to scheduling partial jobs in any time slot. It follows that by considering small enough time slots, the value of an optimal solution to the LP will be no less than the value of the optimal solution to the ILP. After considering this relaxation and taking the dual, we get the dual program shown in Figure 2b.

The complementary slackness conditions of our primal-dual program are

$$\alpha_j > 0 \quad \Rightarrow \quad \sum_{t=r_j}^{T}\sum_{i=1}^{k} x_{jti}\cdot s_i = p_j, \tag{1}$$

$$\beta_t > 0 \quad \Rightarrow \quad \sum_{j\in\mathcal{J}}\sum_{i=1}^{k} x_{jti} = 1, \tag{2}$$

$$x_{jti} > 0 \quad \Rightarrow \quad \beta_t = \alpha_j s_i - P_i - s_i d_j(t - r_j + 1/2) \quad . \tag{3}$$

By complementary slackness, any pair of feasible primal-dual solutions that fulfills these conditions is optimal. We will use this in the following to find a simple way to characterize optimal schedules.

A simple but important observation is that we can write the last complementary slackness condition as $\beta_t = s_i\big(\alpha_j - d_j(t - r_j + \frac{1}{2})\big) - P_i$. Using the complementary slackness conditions, the function $t \mapsto \alpha_j - d_j(t - r_j)$ can be used to characterize optimal schedules. The following definitions capture a parametrized version of these job-dependent functions and state how they imply a corresponding (not necessarily feasible) schedule.

▶ **Definition 1** (Dual Lines and Upper Envelope)**.** For a value $a \geq 0$ and a job j we denote the linear function $D_j^a\colon [r_j,\infty) \to \mathbb{R}, t \mapsto a - d_j(t - r_j)$ as the *dual line* of j with offset a.

Given a job set $H \subseteq \mathcal{J}$ and corresponding dual lines $D_j^{a_j}$, we define the *upper envelope* of H by the upper envelope of its dual lines. That is, the upper envelope of H is a function

$\mathrm{UE}_H\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, t \mapsto \max_{j\in H}\big(D_j^{a_j}(t), 0\big)$. We omit the job set from the index if it is clear from the context.

For technical reasons, we will have to consider the discontinuities in the upper envelope separately.

▶ **Definition 2** (Left Upper Envelope and Discontinuity)**.** Given a job set $H \subseteq \mathcal{J}$ and upper envelope of H, UE_H, we define the *left upper envelope* at a point t as the limit of UE_H as we approach t from the left. That is, the left upper envelope of H is a function $\mathrm{LUE}_H\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, t \mapsto \lim_{t'\to t^-} \mathrm{UE}_H(t')$. Note that an equivalent definition of the left upper envelope is $\mathrm{LUE}_H(t) = \max_{j\in H: r_j < t}\big(D_j^{a_j}(t), 0\big)$.

We say that a point t is a *discontinuity* if UE has a discontinuity at t. Note that this implies that $\mathrm{UE}(t) \neq \mathrm{LUE}(t)$.

For the following definition, let us denote $C_i \coloneqq \frac{P_i - P_{i-1}}{s_i - s_{i-1}}$ for $i \in [k+1]$ as the i-th *speed threshold*. We use it to define the speeds at which jobs are to be scheduled. It will also be useful to define $\hat{C}(x) = \min_{i\in[k+1]}\{\, C_i \mid C_i > x \,\}$ and $\check{C}(x) = \max_{i\in[k+1]}\{\, C_i \mid C_i \leq x \,\}$.

▶ **Definition 3** (Line Schedule)**.** Consider dual lines $D_j^{a_j}$ for all jobs. The corresponding *line schedule* schedules job j in all intervals $I \subseteq [r_j, \infty)$ of maximal length in which j's dual line is on the upper envelope of all jobs (i.e., $\forall t \in I\colon D_j^{a_j}(t) = \mathrm{UE}(t)$). The speed of a job j scheduled at time t is s_i, with i such that $C_i = \check{C}(D_j^{a_j}(t))$.

See Figure 1 for an example of a line schedule. Together with the complementary slackness conditions, we can now easily characterize optimal line schedules.

▶ **Lemma 4.** *Consider dual lines $D_j^{a_j}$ for all jobs. The corresponding line schedule is optimal with respect to fractional weighted flow plus energy if it schedules exactly p_j units of work for each job j.*

Proof. Consider the solution x to the ILP induced by the line schedule. We use the offsets a_j of the dual lines to define the dual variables $\alpha_j \coloneqq a_j + \frac{1}{2}d_j$. For $t \in \mathbb{N}$, set $\beta_t \coloneqq 0$ if no job is scheduled in the t-th slot and $\beta_t \coloneqq s_i D_j^{\alpha_j}(t) - P_i$ if job j is scheduled at speed s_i during slot t. It is easy to check that x, α, and β are feasible and that they satisfy the complementary slackness conditions. Thus, the line schedule must be optimal. ◀

6 Computing an Optimal Schedule

In this section, we describe and analyze the algorithm for computing an optimal schedule. We introduce the necessary notation and provide a formal definition of the algorithm in Subsection 6.1. Then, in Subsection 6.2, we prove the correctness of the algorithm.

6.1 Preliminaries and Formal Algorithm Description

Before formally defining the algorithm, we have to introduce some more notation.

▶ **Definition 5** (Interval Notation)**.** Let $\hat{r}_1, \ldots, \hat{r}_n$ denote the n release times in non-decreasing order. We define Ψ_j as a set of indices with $q \in \Psi_j$ if and only if job j is run between $\hat{r}_q$ and $\hat{r}_{q+1}$ (or after $\hat{r}_n$ for $q = n$). Further, let $x_{\ell,q,j}$ denote the time that the interval corresponding to q begins and $x_{r,q,j}$ denote the time that the interval ends. Let $s_{\ell,q,j}$ denote the speed at which j is running at the left endpoint corresponding to q and $s_{r,q,j}$ denote the speed j is running at the right endpoint. Let $q_{\ell,j}$ be the smallest and $q_{r,j}$ be the largest indices of Ψ_j, i.e., the indices of the first and last execution intervals of j.

Let the indicator variable $y_{r,j}(q)$ denote whether $x_{r,q,j}$ occurs at a release point. Similarly, $y_{\ell,j}(q) = 1$ if $x_{\ell,q,j}$ occurs at r_j, and 0 otherwise. Lastly, $\chi_j(q)$ is 1 if q is not the last interval in which j is run, and 0 otherwise.

We define $\rho_j(q)$ to be the last interval of the uninterrupted block of intervals starting at q, i.e., for all $q' \in \{ q+1, \ldots, \rho_j(q) \}$, we have that $q' \in \Psi_j$ and $x_{r,q'-1,j} = x_{\ell,q',j}$, and either $\rho_j(q) + 1 \notin \Psi_j$ or $x_{r,\rho_j(q),j} \neq x_{\ell,\rho_j(q)+1,j}$.

Within iteration i of the algorithm, τ will represent how much we have raised α_i. We can think of τ as the time parameter for this iteration of the algorithm (not time as described in the original problem description, but time with respect to raising dual-lines). To simplify notation, we do not index variables by the current iteration of the algorithm. In fact, note that every variable in our description of the algorithm may be different at each iteration of the algorithm, e.g., for some job j, $\alpha_j(\tau)$ may be different at the i-th iteration than at the $(i+1)$-st iteration. To further simplify notation, we use D_j^τ to denote the dual line of job j with offset $\alpha_j(\tau)$. Similarly, we use UE^τ to denote the upper envelope of all dual lines D_j^τ for $j \in [i]$ and S_i^τ to denote the corresponding line schedule. As the line schedule changes with τ, so does the set of intervals corresponding to it, therefore we consider variables relating to intervals to be functions of τ as well (e.g., $\Psi_j(\tau)$, $x_{\ell,q,j}(\tau)$, etc.). Prime notation generally refers to the rate of change of a variable with respect to τ, e.g., $\alpha_j'(\tau_0)$ is the rate of change of α_j with respect to τ at τ_0. To lighten notation, we drop τ from variables when its value is clear from the context.

We start by formally defining a relation capturing the idea of jobs affecting each other while being raised.

▶ **Definition 6** (Affection)**.** Consider two different jobs j and j'. We say job j *affects* job j' at time τ if raising (only) the dual line D_j^τ would decrease the processing time of j' in the corresponding line schedule.

We write $j \to j'$ to indicate that j affects j' (and refer to the parameter τ separately, if not clear from the context). Similarly, we write $j \not\to j'$ to state that j does not affect j'.

The affection relation naturally defines a graph on the jobs, which we define below. The following definition assumes that we are in iteration i of the algorithm.

▶ **Definition 7** (Affection Tree)**.** Let $G_i(\tau)$ be the directed graph induced by the affection relation on jobs $1, \ldots, i$. Then the *affection tree* is an undirected graph $A_i(\tau) = (V_i(\tau), E_i(\tau))$ where $j \in V_i(\tau)$ if and and only if j is reachable from i in $G_i(\tau)$, and for $j_1, j_2 \in V_i(\tau)$ we have $(j_1, j_2) \in E_i(\tau)$ if and only if $j_1 \to j_2$ or $j_2 \to j_1$.

Lemma 9 states that the affection tree is indeed a tree. We will assume that $A_i(\tau)$ is rooted at i and use the notation $(j, j') \in A_i(\tau)$ to indicate that j' is a child of j.

Given this notation, we now define four different types of events which intuitively represent the situations in which we must change the rate at which we are raising the dual line. We assume that from τ until an event we raise each dual line at a constant rate. More formally, we fix τ and for $j \in [i]$ and $u \geq \tau$ let $\alpha_j(u) = \alpha_j(\tau) + (u - \tau)\alpha_j'(\tau)$.

▶ **Definition 8** (Event)**.** For $\tau_0 > \tau$, we say that an *event occurs at* τ_0 if there exists $\epsilon > 0$ such that at least one of the following holds for all $u \in (\tau, \tau_0)$ and $v \in (\tau_0, \tau_0 + \epsilon)$:

- The affection tree changes, i.e., $A_i(u) \neq A_i(v)$. This is called an *affection change event.*
- The speed at the border of some interval of some job changes. That is, there exists $j \in [i]$ and $q \in \Psi_j(\tau)$ such that either $s_{\ell,q,j}(u) \neq s_{\ell,q,j}(v)$ or $s_{r,q,j}(u) \neq s_{r,q,j}(v)$. This is called a *speed change event.*

- The last interval in which job i is run changes from ending before the release time of some other job to ending at the release time of that job. That is, there exists a $j \in [i-1]$ and a $q \in \Psi_i(\tau)$ such that $x_{r,q,i}(u) < r_j$ and $x_{r,q,i}(v) = r_j$. This is called a *simple rate change event*.
- Job i completes enough work, i.e., $p_i(u) < p_i < p_i(v)$. This is called a *job completion event*.

A formal description of the algorithm can be found in Algorithm 1.

```
1  for each job i from 1 to n:
2      while p_i(τ) < p_i:                    {job i not yet fully processed in current schedule}
3          for each job j ∈ A_i(τ):
4              calculate δ_{j,i}(τ)           {see Equation (5)}
5          let Δτ be the smallest Δτ returned by any of the subroutines below:
6              (a) JobCompletion(S(τ), i, [α'_1, α'_2, ..., α'_i])      {time to next job completion}
7              (b) AffectionChange(S(τ), A_i(τ), [α'_1, α'_2, ..., α'_i])   {time to next affection change}
8              (c) SpeedChange(S(τ), [α'_1, α'_2, ..., α'_i])           {time to next speed change}
9              (d) RateChange(S(τ), i, [α'_1, α'_2, ..., α'_i])         {time to next rate change}
10         for each job j ∈ A_i(τ):
11             raise α_j by Δτ · δ_{j,i}
12         set τ = τ + Δτ
13         update A_i(τ) if needed            {only if Case (b) returns the smallest Δτ}
```

Algorithm 1 The algorithm for computing an optimal schedule.

6.2 Correctness of the Algorithm

In this subsection we focus on proving the correctness of the algorithm. Throughout this subsection, we assume that the iteration and value of τ are fixed. The following lemma states that A_i is indeed a tree. This structure will allow us to easily compute how fast to raise the different dual lines of jobs in A_i (as long as the connected component does not change).

▶ **Lemma 9.** *Let A_i be the (affection) graph of Definition 7. Then A_i is a tree, and if we root A_i at i, then for any parent and child pair $(\iota_j, j) \in G$ there holds that $d_{\iota_j} < d_j$.*

Recall that we have to raise the dual lines such that the total work done for any job $j \in [i-1]$ is preserved. To calculate the work processed for j in an interval, we must take into account the different speeds at which j is run in that interval. Note that the intersection of j's dual line with the i-th speed threshold C_i occurs at $t = \frac{\alpha_j - C_i}{d_j} + r_j$. Therefore, the work done by a job $j \in [i]$ is given by

$$\begin{aligned} p_j = & \sum_{q \in \Psi_j} s_{\ell,q,j} \left(\frac{\alpha_j - \check{C}(D_j^\tau(x_{\ell,q,j}))}{d_j} + r_j - x_{\ell,q,j} \right) \\ & + \sum_{k : s_{\ell,q,j} > s_k > s_{r,q,j}} s_k \left(\frac{\alpha_j - C_k}{d_j} + r_j - \left(\frac{\alpha_j - C_{k+1}}{d_j} + r_j \right) \right) \\ & + s_{r,q,j} \left(x_{r,q,j} - \left(\frac{\alpha_j - \hat{C}(D_j^\tau(x_{r,q,j}))}{d_j} + r_j \right) \right). \end{aligned}$$

It follows that the change in the work of job j with respect to τ is

$$p'_j = \sum_{q \in \Psi_j} \left[s_{\ell,q,j} \left(\frac{\alpha'_j}{d_j} - x'_{\ell,q,j} \right) + s_{r,q,j} \left(x'_{r,q,j} - \frac{\alpha'_j}{d_j} \right) \right]. \tag{4}$$

For some child j' of j in A_i, let $q_{j,j'}$ be the index of the interval of Ψ_j that begins with the completion of j'. Recall that D_i^τ is raised at a rate of 1 with respect to τ, and for a parent and child (ι_j, j) in the affection tree, the rate of change for α_j with respect to α_{ι_j} used by the algorithm is:

$$\delta_{j,\iota_j} := \Bigg(1 + y_{\ell,j}(q_{\ell,j})\frac{d_j - d_{\iota_j}}{d_j}\frac{s_{\ell,q_{\ell,j},j} - s_{r,\rho_j(q_{\ell,j}),j}}{s_{r,q_{r,j},j}} + \sum_{(j,j')\in A_i}\left((1-\delta_{j',j})\frac{d_j - d_{\iota_j}}{d_{j'} - d_j}\frac{s_{\ell,q_{j,j'},j}}{s_{r,q_{r,j},j}} + \frac{d_j - d_{\iota_j}}{d_j}\frac{s_{\ell,q_{j,j'},j} - s_{r,\rho(q_{j,j'}),j}}{s_{r,q_{r,j},j}}\right)\Bigg)^{-1}. \tag{5}$$

Lemma 12 states that these rates are work-preserving for all jobs $j \in [i-1]$. Note that the algorithm actually uses $\delta_{j,i}$ which we can compute by taking the product of the $\delta_{k,k'}$ over all edges (k,k') on the path from j to i. Similarly we can compute $\delta_{j,j'}$ for all $j, j' \in A_i$.

▶ **Observation 10.** *Since, by Lemma 9, parents in the affection tree are always of lower-density than their children, and since dual lines are monotonically decreasing, we have that $\delta_{\iota_j,j} \leq 1$. Therefore, intersection points on the upper envelope can never move towards the right as τ gets increased.*

The following lemma states how fast the borders of the various intervals change with respect to the change in τ.

▶ **Lemma 11.** *Consider any job $j \in A_i$ whose dual line gets raised at a rate of $\delta_{j,i}$.*

(a) *For an interval $q \in \Psi_j$, if $y_{\ell,j}(q) = 1$, then $x'_{\ell,q,j} = 0$.*

(b) *For an interval $q \in \Psi_j$, if $\chi_j(q) = 1$, then $x'_{r,q,j} = 0$.*

(c) *Let (j,j') be an edge in the affection tree and let q_j and $q_{j'}$ denote the corresponding intervals for j and j'. Then, $x'_{\ell,q_j,j} = x'_{r,q_{j'},j'} = -\frac{\alpha'_j - \alpha'_{j'}}{d_{j'} - d_j}$. Note that this captures the case $q \in \Psi_{j'}$ with $\chi_{j'}(q) = 0$ and $j' \neq i$.*

(d) *For an interval $q \in \Psi_i$, if $\chi_i(q) = 0$, then $x'_{r,q,i} = 0$ or $x'_{r,q,i} = 1/d_i$.*

Equation (4) defines a system of differential equations. In the following, we first show how to compute a work-preserving solution for this system (in which $p'_j = 0$ for all $j \in [i-1]$) if $\alpha'_i = 1$, and then show that there is only a polynomial number of events and that the corresponding τ values can be easily computed.

▶ **Lemma 12.** *For a parent and child $(\iota_j, j) \in A_i$, set $\alpha'_j = \delta_{j,\iota_j}\alpha'_{\iota_j}$, and for $j' \notin A_i$ set $\alpha_{j'} = 0$. Then $p'_j = 0$ for $j \in [i-1]$.*

Although it is simple to identify the next occurrence of job completion, speed change, or simple rate change events, it is more involved to identify the next affection change event. Therefore, we provide the following lemma to account for this case.

▶ **Lemma 13.** *An affection change event occurs at time τ_0 if and only if at least one of the following occurs.*

(a) *An intersection point t between a parent and child $(j,j') \in A_i$ becomes equal to r_j. That is, at $\tau_0 > \tau$ such that $D_j^{\tau_0}(r_j) = D_{j'}^{\tau_0}(r_j) = \mathrm{UE}^{\tau_0}(r_j)$.*

(b) *Two intersection points t_1 and t_2 on the upper envelope become equal. That is, for $(j_1,j_2) \in A_i$ and $(j_2,j_3) \in A_i$, at $\tau_0 > \tau$ such that there is a t with $D_{j_1}^{\tau_0}(t) = D_{j_2}^{\tau_0}(t) = D_{j_3}^{\tau_0}(t) = \mathrm{UE}^{\tau_0}(t)$.*

(c) *An intersection point between j and j' meets the (left) upper envelope at the right endpoint of an interval in which j' was being run. Furthermore, there exists $\epsilon > 0$ so that for all $\tau \in (\tau_0 - \epsilon, \tau_0)$, j' was not in the affection tree.*

6.2.1 The Subroutines

Recall that there are four types of events that cause the algorithm to recalculate the rates at which it is raising the dual lines. In Lemma 13 we gave necessary and sufficient conditions for affection change events to occur. The conditions for the remaining event types to occur follow easily from Lemma 11 and Observation 10. Given the rates at which the algorithm is raising the dual lines, we can then easily calculate the time until each of these events will occur next. The subroutines describing these calculations are left for the full version.

6.2.2 Completing the Correctness Proof

We are now ready to prove the correctness of the algorithm. Note that we handle termination in Theorem 15, where we prove a polynomial running time for our algorithm.

▶ **Theorem 14.** *Assuming that Algorithm 1 terminates, it computes an optimal schedule.*

Proof. The algorithm outputs a line schedule S, so by Lemma 4, S is optimal if for all jobs j the schedule does exactly p_j work on j. We now show that this is indeed the case.

For a fixed iteration i, we argue that a change in the rate at which work is increasing for j (i.e., a change in p'_j) may occur only when an event occurs. This follows from Equation (4), since the rate only changes when there is a change in the rate at which the endpoints of intervals move, when there is a change in the speed levels employed in each interval, or when there is an affection change (and hence a change in the intervals of a job or a change in α'_j). These are exactly the events we have defined. It can be shown that the algorithm recalculates the rates at any event (proofs deferred to the full version), and by Lemma 12 it calculates the correct rates such that $p'_j(\tau) = 0$ for $j \in [i-1]$ and for every τ until some τ_0 such that $p_i(\tau_0) = p_i$, which the algorithm calculates correctly (proof also deferred to the full version). Thus we get the invariant that after iteration i we have a line schedule for the first i jobs that does p_j work for every job $j \in [i]$. The theorem follows. ◀

7 The Running Time

The purpose of this section is to prove the following theorem.

▶ **Theorem 15.** *Algorithm 1 takes $O(n^4k)$ time.*

We do this by upper bounding the number of events that can occur. This is relatively straightforward for job completion, simple rate change, and speed change events, which can occur $O(n)$, $O(n^2)$, and $O(n^2k)$ times, respectively. However, bounding the number of times an affection change event can occur is more involved: One can show that whenever an edge is removed from the affection tree, there exists an edge which will never again be in the affection tree. This implies that the total number of affection change events is upper bounded by $O(n^2)$ as well. It can be shown that the next event can always be calculated in $O(n^2)$ time, and that the affection tree can be updated in $O(n)$ time after each affection change event. By combining these results it follows that our algorithm has a running time of $O(n^4k)$.

Due to space constraints, the missing proofs in the statements above are left for the full version.

References

1 Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

2 Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

3 Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

4 Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *Proceedings of the 35th International Conference on Automata, Languages, and Programming (ICALP)*, pages 409–420, 2008.

5 Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2), 2013.

6 Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

7 Neal Barcelo, Daniel Cole, Dimitrios Letsios, Michael Nugent, and Kirk Pruhs. Optimal energy trade-off schedules. *Sustainable Computing: Informatics and Systems*, 3:207–217, 2013.

8 Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Non-clairvoyant speed scaling for weighted flow time. In *Proceedings of the 18th annual European Symposium on Algorithms (ESA), Part I*, pages 23–35, 2010.

9 Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1123–1140, 2014.

10 Jacques Labetoulle, Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In Pulleyblank H. R., editor, *Progress in combinatorial optimization*, pages 245–261. Academic Press, 1984.

11 Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the 16th annual European Symposium on Algorithms (ESA)*, pages 647–659, 2008.

12 Nicole Megow and José Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming (ICALP) - Volume Part I*, pages 745–756, 2013.

13 Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.