Algorithmica manuscript No.
(will be inserted by the editor)

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

Antonios Antoniadis · Neal Barcelo · Mario Consuegra · Peter Kling · Michael Nugent · Kirk Pruhs · Michele Scquizzato

Received: date / Accepted: date

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.

Keywords Scheduling · Flow time · Energy efficiency · Speed scaling · Primal-dual

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

A preliminary version of this work appeared in the Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS), 2014 (see [4]).

A. Antoniadis
Max-Planck Institut für Informatik, Saarbrücken, Germany
E-mail: aantonia@mpi-inf.mpg.de

N. Barcelo · M. Nugent · K. Pruhs
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

M. Consuegra
Google (work done while at Florida International University)
E-mail: marioecd@google.com

P. Kling
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
E-mail: pkling@sfu.ca

M. Scquizzato
Department of Computer Science, University of Houston, Houston, TX, USA
E-mail: michele@cs.uh.edu

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. Each mode has a different speed and power consumption, with the higher speed modes being 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 consumption. Our main result is a polynomial time algorithm to compute such an optimal trade-off schedule.

Intuition and Significance of Trade-off Schedules. We want to informally elaborate on the statement of our main result. Fully formal definitions are given 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:

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 corresponding 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 the job that is being processed and the mode of the processor at any point in time.

Our Performance Measure: 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 its execution 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.)

Outline of the Article. In Section 2 we explain the relationship of our result to related results in the literature. In Section 3 we introduce a formal model and notation. 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. Section 6 formalizes this structural information we gain from the primal-dual interpretation. In Section 7 we use this information to develop our algorithm. Finally, in Section 8 we analyze the running time of our algorithm and finish with a short conclusion in Section 9.

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, 15] give polynomial time algorithms for the special case of our problem where the densities of all units of work are the same. In [15], Pruhs et al. give 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. Albers and Fujiwara [2] give a dynamic programming algorithm and deserve credit for introducing the notion of trade-off schedules. Barcelo et al. [8] give a polynomial-time algorithm for recognizing an optimal schedule. They 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, [8] was not able to provide any bound, even exponential, on the time of this algorithm, nor was [8] able to provide any way to discretize this algorithm.

To reemphasize, the prior literature [2, 8, 15] 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 [8] 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. 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.

In a recent work, Barcelo et al. [9] map out the problem space that results from varying some of our problem parameters. More specifically, they give polynomial-time algorithms and NP-hardness results for various formulations of the speed scaling problem, depending on how one models the processor (e.g., discrete vs. continuous speeds or the nature of relationship between speed and power), the performance objective (e.g., whether jobs are of equal or unequal importance, and whether one is interested in minimizing waiting times of jobs or of work), and how one handles the dual objective (e.g., whether they are combined in a single objective, or whether one objective is transformed into a constraint).

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 its work 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 [12], or from the fact that minimizing total weighted flow, without release times, subject to an energy budget is NP-hard [14].

Given the above mentioned hardness result, it is not surprising that there is a fair number of papers that study approximately computing optimal trade-off schedules, both offline and online. [14] also gives PTASs for minimizing total flow without release times subject to an energy budget in both the continuous and discrete speed settings. [2, 3, 5–7, 10, 11, 13] consider online algorithms for optimal total flow and energy, [5, 7] consider online algorithms for fractional flow and energy. See the survey by Albers [1] for a more detailed overview of energy-efficient algorithms.

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 $0 < s_1 < s_2 < \cdots < s_k$. The power consumption of the processor while running at speed s_i is $P_i \geq 0$. We use $\mathcal{S} \coloneqq \{ s_1, s_2, \ldots, s_k \}$ to denote the set of speeds and $\mathcal{P} \coloneqq \{ P_1, P_2, \ldots, P_k \}$ 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. For each time t, a schedule S must decide which job to process and at what speed. Preemption is allowed, that is, a job may be suspended at any point in time and resumed later on. We model a schedule S by a (measurable) 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 $I_j = J^{-1}(j) \cap \left[r_j, \infty\right)$ be the set of times during which it is processed. A feasible schedule must finish the work of all jobs. That is, the inequality $\int_{I_j} V(t)\,\mathrm{d}t \geq p_j$ 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 $v_j(t)$ denotes the work of job j that is processed at time t (i.e., $v_j(t) = V(t)$ if $J(t) = j$, and $v_j(t) = 0$ otherwise), the fractional flow time of job j is $\widetilde{F}_j \coloneqq \int_{r_j}^\infty (t - r_j) \frac{v_j(t)}{p_j}\,\mathrm{d}t$. 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}. Most of the time, our analysis assumes all densities to be distinct. This is without loss of generality, and we explain at the end of our analysis section (see Section 7.3) how the algorithm can be changed to handle jobs of equal densities.

4 Overview

In this section we give an overview of our algorithm design and analysis. We first outline how we extract geometric information from the primal-dual formulation of the problem, and then give an example of this geometric information, providing insight into how this yields an optimality condition. Finally, we give a first overview of how to leverage this condition when designing our algorithm.

From Primal-Dual to Dual Lines. We start by considering a natural linear programming formulation of the problem. We then consider the dual linear

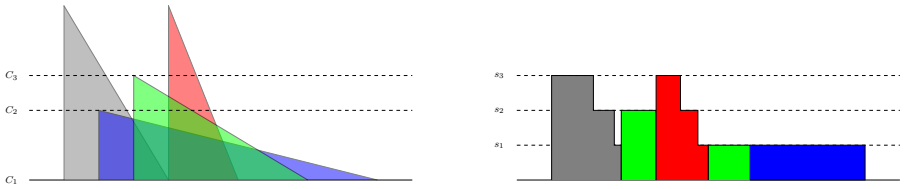


Fig. 1: The dual lines for a 4-job instance, and the associated schedule.

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 [8], 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 a *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.

Example Schedule & Dual Lines. 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.

Optimality Condition. 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 satisfying this property.

Algorithmic Idea. 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 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 it is 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 previously added 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 (a) the affection graph is a tree, (b) this affection tree is correctly computed, (c) the four aforementioned events are exactly the ones that change the affection tree, and (d) 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

This section derives the structural optimality condition (Theorem 1) on which our algorithm is based. We do so by means of an integer linear programming (ILP) description of our problem. Before we give the ILP and derive the condition, we show how to divide time into discrete time slots with certain properties.

Discretizing Time. We define time slots in which the processor runs at constant speed and processes at most one job. Note that, in general, these time slots may be arbitrarily small, yielding an ILP with many variables. At first glance, this seems problematic, as it renders a direct solution approach less attractive. However, we are actually not interested in solving the resulting ILP directly. Instead, we merely strive to use it and its dual in order to obtain some simple structural properties of an optimal schedule.

$$\min \quad \sum_{j\in\mathcal{J}} \sum_{t=r_j}^{T} \sum_{i=1}^{k} x_{jti}\big(P_i + s_i d_j (t - r_j + 1/2)\big)$$

$$\text{s.t.} \quad \sum_{t=r_j}^{T} \sum_{i=1}^{k} x_{jti} \cdot s_i \geq p_j \qquad \forall j$$

$$\sum_{j\in\mathcal{J}} \sum_{i=1}^{k} x_{jti} \leq 1 \qquad \forall t$$

$$x_{jti} \in \{0, 1\} \quad \forall j, t, i$$

(a) ILP formulation of our problem.

$$\max \quad \sum_{j\in\mathcal{J}} p_j \alpha_j - \sum_{t=1}^{T} \beta_t$$

$$\text{s.t.} \quad \beta_t \geq \alpha_j s_i - P_i - s_i d_j (t - r_j + 1/2) \quad \forall j, t, i : t \geq r_j$$

$$\alpha_j \geq 0 \quad \forall j$$

$$\beta_t \geq 0 \quad \forall t$$

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

Fig. 2: Primal-dual formulation of our problem.

To this end, consider $\varepsilon > 0$ and let $T \in \mathbb{N}$ be such that $T\varepsilon$ is an upper bound on the completion time of non-trivial[1] schedules (e.g., $T\varepsilon \geq \max_j(r_j + \sum_j p_j/s_1)$). Given a fixed problem instance, there is only a finite number of jobs and, without loss of generality, an optimal schedule performs only a finite number of speed switches and preemptions[2]. Thus, we can choose $\varepsilon > 0$ such that

(a) any release time r_j is a multiple of ε,
(b) an optimal schedule can use constant speed during $\big[(t-1)\varepsilon, t\varepsilon\big)$, and
(c) there is at most one job processed during $\big[(t-1)\varepsilon, t\varepsilon\big)$.

We refer to an interval $\big[(t-1)\varepsilon, t\varepsilon\big)$ as the t-th time slot. By rescaling the problem instance we can assume that time slots are of unit size (scale r_j by $1/\varepsilon$ and scale s_i as well as P_i by ε).

ILP & Dual Program. Let the indicator variable x_{jti} denote whether job j is processed in slot t at speed s_i. Note that T as defined above is an 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 an optimal schedule can use highest

[1] A non-trivial schedule is one that never runs at speed 0 when there is work remaining.

[2] For the finite number of preemptions, note that any schedule can be transformed to a highest density first (HDF) schedule (see also next paragraph), in which the number of preemptions is bounded by the number of release times. For the finite number of speed switches, consider the average speed s in any maximal execution interval of some job j (where it runs at possible different speeds). If s lies between discrete speeds s_i and s_{i+1}, the schedule can be changed to run first for some time at speed s_{i+1} and then for some time at speed s_i without increasing the energy cost or the flow time.

density first (HDF) as its scheduling policy[3]. 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

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

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

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

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.

Dual Lines. 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 parameterized version of these job-dependent functions and state how they imply a corresponding (not necessarily feasible) schedule.

Definition 1 (Dual Lines & Upper Envelope) For a value $a \geq 0$ and a job j we define 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 the 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 & 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 the 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 $\mathrm{UE}(t) \neq \mathrm{LUE}(t)$.

[3] Given any non-HDF schedule, one can swap two (arbitrarily small) portions of jobs that violate HDF. By definition of the cost function, this results in a strictly better schedule.

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.

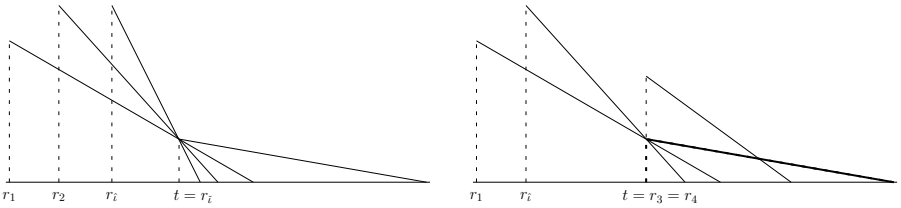
Theorem 1 *Consider dual lines $D_j^{a_j}$ for all jobs. The corresponding line schedule is optimal with respect to fractional weighted flow plus energy if it schedules exactly p_j units of work for each job j.*

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 Affection & Raising Process

Recall the algorithmic idea sketched in Section 4: We aim to develop a primal-dual algorithm that raises dual variables in a structured fashion. Theorem 1 provides us with some motivation for how to organize this raising. A first approach might be to raise the dual line of a new job i, leaving the dual lines of previously scheduled jobs untouched. At some point, its dual line will claim enough time on the upper envelope to be fully processed. However, in doing so we may affect (i.e., reduce) the time windows of other (already scheduled) jobs. Thus, while raising i's dual line, we must keep track of any affected jobs and ensure that they remain fully scheduled. This section formalizes this idea by defining affections and by structuring them in such a way that we can efficiently keep track of them.

Notation for the Raising Process. 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



(a) Affections: $\{1, 2, \hat{\imath}\} \to \{\check{\imath}\}$, $\{1, 2\} \to \{\hat{\imath}\}$, and $\{1, 2, \hat{\imath}, \check{\imath}\} \not\to \{1, 2\}$.

(b) Affections: $1 \to \hat{\imath}$ and $\{1, \hat{\imath}, 3\} \not\to \{1, 3\}$. (Job 3 is denoted by the thick line.)

Fig. 3: Illustration of (a) Observation 3 and (b) Observation 4.

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. Prime notation generally refers to the rate of change of a variable with respect to τ. To lighten notation further, we drop τ from variables when its value is clear from the context.

6.1 Affected Jobs

Let us define a relation capturing the idea of jobs affecting each other while being raised.

Definition 4 (Affection) Consider two different jobs j and j'. We say that 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'. Before we show how to structure the affection between different jobs, let us collect some simple observations on when and how jobs can actually affect each other. We start with observing that jobs can affect each other only if their dual lines intersect on the upper envelope (Figure 3(a)) or left upper envelope (Figure 3(b)).

Observation 2 *Given jobs j and j' with $j \to j'$, their dual lines must intersect on the upper envelope, or on the left upper envelope at a discontinuity. That is, if t is the intersection point of j and j', we have either $D_j^\tau(t) = D_{j'}^\tau(t) = \mathrm{UE}^\tau(t)$, or $D_j^\tau(t) = D_{j'}^\tau(t) = \mathrm{LUE}^\tau(t)$ and t is a discontinuity. Further there must be some $\epsilon > 0$ such that j' is run in either $(t-\epsilon, t)$ or $(t, t+\epsilon)$.*

The following two observations are the counterpart of Observation 2. More precisely, Observation 3 states that only the most and least dense jobs intersecting at the upper envelope can be affected (Figure 3(a)). Similarly, Observation 4 states that only the highest density job intersecting at the left upper envelope can be affected (Figure 3(b)).

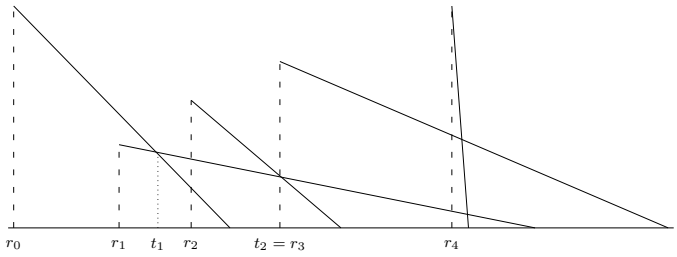


Fig. 4: Observation 6. Both the upper envelope and the left upper envelope cases: note that $D_0^\tau(t') < \mathrm{LUE}^\tau(t') \leq \mathrm{UE}^\tau(t')$ for all $t' > t_1$, and $D_2^\tau(t') < \mathrm{LUE}^\tau(t') \leq \mathrm{UE}^\tau(t')$ for all $t' > t_2$.

Observation 3 *For $t \in \mathbb{R}_{\geq 0}$ consider the maximal set H_t of jobs that intersect the upper envelope at t and define $H_t^- \coloneqq H_t \cap \{\, j \mid r_j < t \,\}$. Let $\check{\imath} \in H_t$ denote the job of lowest density and let $\hat{\imath} \in H_t^-$ denote the job of highest density in the corresponding sets (assuming the sets are nonempty). The following hold:*

(a) For all $j \in H_t \setminus \{\, \check{\imath} \,\}$ we have $j \to \check{\imath}$.
(b) For all $j \in H_t^- \setminus \{\, \hat{\imath} \,\}$ we have $j \to \hat{\imath}$.
(c) For all $j \in H_t$ and $j' \in H_t \setminus \{\, \check{\imath}, \hat{\imath} \,\}$ we have $j \not\to j'$.

Observation 4 *For $t \in \mathbb{R}_{\geq 0}$ consider the maximal set H_t of jobs that intersect the left upper envelope at t where t is a discontinuity. Define $H_t^- \coloneqq H_t \cap \{\, j \mid r_j < t \,\}$. Let $\hat{\imath} \in H_t^-$ denote the job of highest density in H_t^- (assuming it is nonempty). The following hold:*

(a) For all $j \in H_t^- \setminus \{\, \hat{\imath} \,\}$ we have $j \to \hat{\imath}$.
(b) For all $j \in H_t$ and $j' \in H_t \setminus \{\, \hat{\imath} \,\}$ we have $j \not\to j'$.

The final two observations are based on the fact that once the dual lines of a higher density job and a lower density job intersect, the higher density job is "dominated" by the lower density job (Figure 4).

Observation 5 *No job $j \in [i-1]$ can intersect a job j' of lower density at its own release time r_j.*

Observation 6 *Given jobs j and j' with $d_j > d_{j'}$ that intersect on the upper envelope or left upper envelope at point t, we have that $\mathrm{UE}^\tau(t') \geq \mathrm{LUE}^\tau(t') \geq D_{j'}^\tau(t') > D_j^\tau(t')$, for all $t' > t$.*

The following proposition is used to prove several lemmas in the next section. It characterizes a situation in which three jobs intersect at the same point on the (left) upper envelope.

Proposition 1 *Consider three jobs j_1, j_2, j_3 with $j_1 \to j_2$ or $j_2 \to j_1$, and $j_2 \to j_3$ or $j_3 \to j_2$ (we say j_2 is the connecting job of j_1 and j_3). If $d_{j_2} > \max(d_{j_1}, d_{j_3})$, then all three jobs intersect on the upper (or left upper) envelope at the same point t.*

Proof Due to Observation 2, the dual line of j_2 intersects the dual lines of j_1 and j_3 somewhere on the upper (or left upper) envelope. Let t_1 denote the intersection point of j_1 and j_2 on the upper (or left upper) envelope. Similarly, let t_2 denote the intersection point of j_2 and j_3 on the upper (or left upper) envelope. For the sake of a contradiction, assume $t_1 \neq t_2$. Without loss of generality, let $t_1 < t_2$. Since $d_{j_2} > d_{j_1}$, by Observation 6 we have $D^{\tau}_{j_2}(t') < \mathrm{LUE}^{\tau}(t') \leq \mathrm{UE}^{\tau}(t')$ for all $t' > t_1$. In particular, $D^{\tau}_{j_2}(t_2) < \mathrm{LUE}^{\tau}(t_2) \leq \mathrm{UE}^{\tau}(t_2)$. This contradicts Observation 2, which states that j_2 and j_3 must intersect *on* the upper (or left upper) envelope. □

6.2 Structuring the Affection

Equipped with these observations, we provide additional structural properties about how different jobs can affect each other. Assume jobs to be ordered by decreasing density and fix a job i. In the following, we study how the raising of job i can affect the already scheduled jobs in $\{1, 2, \ldots, i-1\}$.

Define level sets $\mathcal{L}_0 \coloneqq \{i\}$ and $\mathcal{L}_l \coloneqq \{j \mid \exists j_- \in \mathcal{L}_{l-1} \colon j_- \to j\} \setminus \bigcup_{l'=0}^{l-1} \mathcal{L}_{l'}$ for an integer $l \geq 1$. That is, a job j is in level set $\mathcal{L}_l$ if and only if the shortest path from i to j in the graph induced by the affection relation is of length l. Lemma 1 shows that densities are monotonously increasing along affection relations. Lemma 2 proves that no two jobs of the same level can affect each other, and Lemma 3 shows that each job can be affected by at most one lower-level job. Finally, Lemma 4 and Lemma 5 show that there are no circles or backward edges in the graph. We use these insights in Lemma 6 to prove that the graph induced by the affection relations forms a tree.

Lemma 1 *Consider two jobs $j_0 \in \mathcal{L}_l$ and $j_+ \in \mathcal{L}_{l+1}$ with $j_0 \to j_+$. Then job j_+ has a larger density than job j_0. That is, $d_{j_+} > d_{j_0}$.*

Proof We prove the statement of the lemma by induction. The base case $l = 0$ is trivial, as i has the lowest density of all jobs and, by construction, $\mathcal{L}_0 = \{i\}$. Now consider the case $l \geq 1$ and let $j_- \in \mathcal{L}_{l-1}$ be such that $j_- \to j_0$. By the induction hypothesis, we have $d_{j_0} > d_{j_-}$ (remember that, w.l.o.g., densities are assumed to be distinct; cf. Section 7.3). For the sake of a contradiction, assume $d_{j_+} < d_{j_0}$. By Proposition 1, all three jobs intersect on the upper (or left upper) envelope at the same point t. Note that this intersection point must lie on the upper envelope, since otherwise it would lie on the left upper envelope at a discontinuity and, by Observation 4, $j_0 \not\to j_+$. Further, because $j_0 \neq i$ and $d_{j_0} > d_{j_+}$, Observation 5 implies $r_{j_0} < t$. Together with $d_{j_+} < d_{j_0}$ and $j_0 \to j_+$, this implies that j_+ has minimal density among all jobs intersecting the upper envelope in t (by Observation 3). We get $j_- \to j_+$ and, thus, $j_+ \in \mathcal{L}_l$. This contradicts j_+ being a level $l+1$ node. □

Lemma 2 *Given two level l jobs $j_1, j_2 \in \mathcal{L}_l$, we have $j_1 \not\to j_2$ and $j_2 \not\to j_1$.*

Proof The statement is trivial for $l = 0$, as $\mathcal{L}_0 = \{i\}$. For $l \geq 1$ consider $j_1, j_2 \in \mathcal{L}_l$ and assume, for the sake of a contradiction, that $j_1 \to j_2$ (the

case $j_2 \to j_1$ is symmetrical). Let $\iota_1, \iota_2 \in \mathcal{L}_{l-1}$ with $\iota_1 \to j_1$ and $\iota_2 \to j_2$. By Lemma 1 we have $d_{\iota_1} < d_{j_1}$ and $d_{\iota_2} < d_{j_2}$. Let t_0 denote the intersection point of j_1 and j_2, t_1 the intersection point of j_1 and ι_1, and t_2 the intersection point of j_2 and ι_2. Analogously to the proof of Lemma 1, one can see that $t_0 = \min\{t_1, t_2\}$ (as otherwise at least one of these intersection points would not lie on the (left) upper envelope). We distinguish the following cases:

Case $t_0 = t_1 < t_2$: First note that t_1 and t_0 cannot lie on the left upper envelope at a discontinuity, since by Observation 4 either $j_1 \not\to j_2$ or $\iota_1 \not\to j_1$. So, by Observation 2, t_0 and t_1 lie on the upper envelope. Job j_2 must have minimal density among all jobs intersecting the upper envelope at t_1, as otherwise its intersection point with ι_2 cannot lie on the upper envelope. But then, by Observation 3, we have $\iota_1 \to j_2$. Together with Lemma 1 this implies $d_{j_2} > d_{\iota_1}$, contradicting the minimality of j_2's density.

Case $t_0 = t_2 < t_1$: By Observation 2 j_1, j_2 and ι_2 either lie on the left upper envelope or the upper envelope. Assume they are on the left upper envelope. Note that since ι_2 and j_1 intersect at t_0, and $t_1 > t_0$, it must be that $\iota_1 \neq \iota_2$ and therefore $l \geq 2$ in this case. Also, since $t_1 > t_0$ is a point where j_1 is on the upper envelope, it must be that j_1 is less dense than ι_2. However this implies that ι_2 is not on the left upper envelope or upper envelope to the right of t_0. Since it is not the root ($l \geq 2$) there must be some point $t < t_0$ such that ι_2 intersects a less dense job on the (left) upper envelope (its parent). This contradicts ι_2 being on the left upper envelope at t_0. If instead j_1, j_2 and ι_2 lie on the upper envelope the same argument used in the first case applies.

Case $t_0 = t_1 = t_2$: The same argument as in the first case shows that these points do not lie on the left upper envelope at a discontinuity but must lie on the upper envelope. Without loss of generality, assume $d_{j_1} > d_{j_2}$. We get $r_{j_1} < t_1$ (Observation 5). With $d_{j_2} > d_{\iota_2}$ and Observation 3 this implies $\iota_2 \not\to j_2$, contradicting the definition of ι_2. □

Lemma 3 *A level l job cannot be affected by more than one job of a lower level.*

Proof The statement is trivial for $l \in \{0, 1\}$. Thus, consider a job $j \in \mathcal{L}_l$ for $l \geq 2$ and let j_1 and j_2 be two different lower level jobs with $j_1 \to j$ and $j_2 \to j$. By definition, both j_1 and j_2 must be level $l-1$ jobs (otherwise, j would be a job of level $< l$). Thus, by Lemma 1, $d_j > \max(d_{j_1}, d_{j_2})$. Applying Proposition 1 yields that all three jobs intersect on the upper envelope or on the left upper envelope (at a discontinuity) at the same point t. Let us first assume they intersect at the upper envelope. Observation 3 implies that j has maximal density of all jobs intersecting the upper envelope at t (as otherwise j can be affected by neither j_1 nor j_2, both having a lower density). Consider the lowest density job $\check{\iota}$ intersecting the upper envelope at t. By Observation 3, at least one among j_1 and j_2 must affect $\check{\iota}$. Assume, without loss of generality, it is j_1. This implies that $\check{\iota}$ has level $l' \leq l$. Actually, we must have $l' < l$, because otherwise $d_{\check{\iota}} < d_{j_1}$ would contradict Lemma 1. Similarly, $l' = l - 1$

would contradict Lemma 2. Thus, we have $l' \leq l-2$. But since we have $\check{\iota} \to j$, we get a contradiction to j being a level l node.

Now assume that all three jobs intersect at a discontinuity of the left upper envelope. Observation 4 tells us that j must be the job of highest density intersecting at t. Assume without loss of generality that $d_{j_1} < d_{j_2}$. Then, by Observation 6, j_2 is not on the (left) upper envelope to the right of t. However, since it is not the root ($l \geq 2$), there must be some job $\check{\iota}$ of smaller density that intersects j_2 on the (left) upper envelope to the left of t (its parent). This contradicts that j_2 would be on the left upper envelope at t. □

Lemma 4 *Consider two jobs $j_1 \in \mathcal{L}_{l_1}$ and $j_2 \in \mathcal{L}_{l_2}$ with $l_2 - l_1 \geq 2$. Then, we must have $j_2 \not\to j_1$.*

Proof For the sake of a contradiction, assume $j_2 \to j_1$. By Observation 2 j_1 and j_2 intersect at a point t on the (left) upper envelope. By Observations 3 and 4, j_1 must have either minimal or maximal density among all jobs that intersect at t on the (left) upper envelope (or we could not have $j_2 \to j_1$). Consider first the case that j_1 has minimal density. In particular, $d_{j_1} < d_{j_2}$. Let j_2' be the job of level $l_2 - 1 \geq 1$ with $j_2' \to j_2$. By Lemma 1 we have $d_{j_2'} < d_{j_2}$. Thus, we can apply Proposition 1 to see that the intersection of j_2' and j_2 is also at t. But then, j_2 must be the job of maximal density that intersects the (left) upper envelope at t (by Observations 3 and 4). Then again we have $j_1 \to j_2$ (also by Observations 3 and 4), which implies that j_2 is of level $l_1 + 1 < l_2$, a contradiction.

So consider the second case, where j_1 has maximal density. In particular, note that j_1 cannot be of level 0 (the only job at level 0 is i, and by choice of our algorithm we have $d_i < d_{j_2} < d_{j_1}$). For $i \in \{1, 2\}$ let j_i' be the level $l_i - 1 \geq 0$ job with $l_i' \to l_i$. Let t_i be the point where j_i and j_i' intersect on the (left) upper envelope. By Observation 6 we have $t_i \geq t$ (otherwise $D^\tau_{j_i}(t) < \mathrm{LUE}^\tau(t) \leq \mathrm{UE}^\tau(t)$, contradicting that j_i is on the (left) upper envelope at t). In fact, for $i = 1$ we have $t_1 = t$ by Proposition 1. Now we consider two subcases. If $t_2 = t$, Observations 3 and 4 imply $j_1' \to j_2$ (j_2 must be the minimal density job intersecting the (left) upper envelope at t) and, thus, that j_2 is of level $l_1 - 1 + 1 = l_1 < l_2$, a contradiction. Otherwise, if $t_2 > t$, there is a job $j' \neq j_2$ of minimal density intersecting the upper envelope at t. In particular, $d_{j'} < d_{j_2}$ and Observation 6 implies that $D^\tau_{j_2}(t_2) < \mathrm{LUE}^\tau(t_2) \leq \mathrm{UE}^\tau(t_2)$, contradicting that j_2 is at the (left) upper envelope at t_2. □

Lemma 5 *Consider two jobs $j_1 \in \mathcal{L}_{l_1}$ and $j_2 \in \mathcal{L}_{l_2}$ with $l_2 = l_1 + 1$, and $j_1 \to j_2$. Then, if there exists a job $j_3 \in \mathcal{L}_{l_1}$ such that $j_2 \to j_3$, it must be that $j_3 = j_1$.*

Proof For the sake of contradiction, assume there exists a job $j_3 \neq j_1$ such that $j_3 \in \mathcal{L}_{l_1}$ and $j_2 \to j_3$. First, note that by Lemma 1 we have $d_{j_1} < d_{j_2}$. Let t_1 be the intersection of j_1 and j_2, and t_2 be the intersection of j_2 and j_3. There are two cases to consider. In the first case, assume $t_1 = t_2$ and note that the

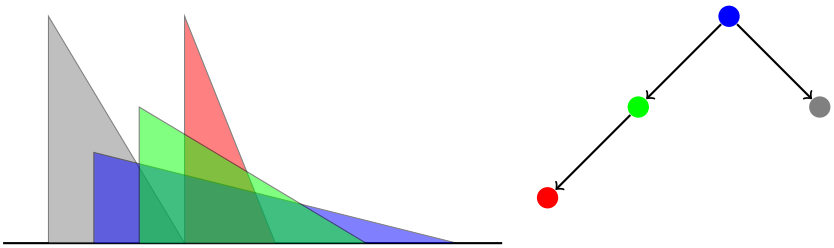


Fig. 5: Affection tree example, rooted at the blue job.

intersection must lie on the upper envelope: if it were on the left upper envelope at a discontinuity, $j_1 \to j_2$ and $j_2 \to j_3$ would contradict Observation 4. Since $d_{j_1} < d_{j_2}$, either j_1 or j_3 is the job with lowest density. Then, since $t_1 = t_2$, by Observation 3 either $j_1 \to j_3$ or $j_3 \to j_1$. Both cases contradict Lemma 2 since $j_1, j_3 \in \mathcal{L}_{l_1}$.

In the second case, assume $t_1 \neq t_2$. If $t_1 < t_2$, then, since $d_{j_1} < d_{j_2}$ by Observation 6, $D^{\tau}_{j_2}(t') < \mathrm{LUE}^{\tau}(t') \leq \mathrm{UE}^{\tau}(t')$ for all $t' > t_1$ which contradicts j_2 being on the (left) upper envelope at t_2. For the case $t_1 > t_2$, a similar argument shows that j_2 must be the least dense job that intersects the upper envelope at t_2, as otherwise $D^{\tau}_{j_2}(t_1) < \mathrm{LUE}^{\tau}(t_1) \leq \mathrm{UE}^{\tau}(t_1)$. If the jobs meet on the upper envelope at t_2, Observation 3 yields $j_3 \to j_2$. Together with $j_1 \to j_2$ and $j_3 \neq j_1$, this contradicts Lemma 3. If the jobs meet on the left upper envelope at t_2, we see similarly to previous proofs that j_3 is not the root, cannot be processed to the right of t_2 (since j_2 is less dense), and its less dense parent muss intersect it to the left of t_2. But then, j_2 cannot be on the left upper envelope at t_2, a contradiction. □

6.3 Affection Tree

We will now formally define and study the graph defined by the affection relation. Using the lemmas from Section 6.2, we will show that this graph is a tree (Lemma 6).

Definition 5 (Affection Tree) Let $G_i(\tau)$ be the directed graph induced by the affection relation on jobs $1, 2, \ldots, i$. The *affection tree* is an undirected graph $A_i(\tau) = (V_i(\tau), E_i(\tau))$ where $j \in V_i(\tau)$ if 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 $j_2 \to j_1$.

See Figure 5 for an illustration of this definition.

Lemma 1 to Lemma 5 imply that, if we omit edge directions, this subgraph indeed forms a tree rooted at i such that all children of a node j are of higher density. We state and prove this in the next lemma.

Lemma 6 *Let A_i be the (affection) graph of Definition 5. Then A_i is a tree, and if we root A_i at i, then for any parent and child pair $(\iota_j, j) \in G$ it holds that $d_{\iota_j} < d_j$.*

Proof Assume, for the sake of contradiction, that there exists a cycle C in G. Let v be a node in C that belongs to the highest level set, say $\mathcal{L}_{l_1}$. Note that such a v is unique since otherwise there would be two nodes in the same level with at least one having an affection to the other contradicting Lemma 2. Let v_1, v_2 be the neighbors of v in C and $v_3 \in \mathcal{L}_{l_1-1}$ be the node such that $v_3 \to v$ (note that it may be $v_3 = v_1$ or $v_3 = v_2$). Note that by Lemma 4 we also have $v_1, v_2 \in \mathcal{L}_{l_1-1}$. By Lemma 3, either $v_1 \not\to v$ or $v_2 \not\to v$. Assume without loss of generality this is v_1. Since v_1 is a neighbor of v in C and $v_1 \not\to v$, we have $v \to v_1$. However, this contradicts Lemma 5. □

For the remainder of the paper, we will always assume $A_i(\tau)$ is rooted at i and use the notation $(j, j') \in A_i(\tau)$ to indicate that j' is a child of j. The proven tree structure of the affection graph 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).

7 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 Section 7.1. In Section 7.2, we prove the correctness of the algorithm. Finally, Section 7.3 explains how the algorithm and the analysis need to be adapted to allow for arbitrary (not pairwise different) densities.

7.1 Preliminaries and Formal Algorithm Description

Our algorithm will use the affection tree to track the jobs affected by the raising of the current job i and compute corresponding raising rates. The raising will continue until job i is completely scheduled, or there is some structural change causing us to recompute the rates at which we are raising dual lines. For example a change in the structure of the affection tree when new nodes are affected will cause us to pause and recompute. The intuition for each event is comparatively simple (see Definition 6), but their formalization is quite technical, requiring us to explicitly label the start and ending points of each single execution interval of each job. To do so, we introduce the following *interval notation*. See Figure 6 for a corresponding illustration.

Interval Notation. Let $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_n$ denote the jobs' 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 $q = n$). Job j must run in a (sub-) interval of $\big[\hat{r}_q, \hat{r}_{q+1}\big)$. Let $x_{-,q,j}$ denote the left and $x_{+,q,j}$ denote the

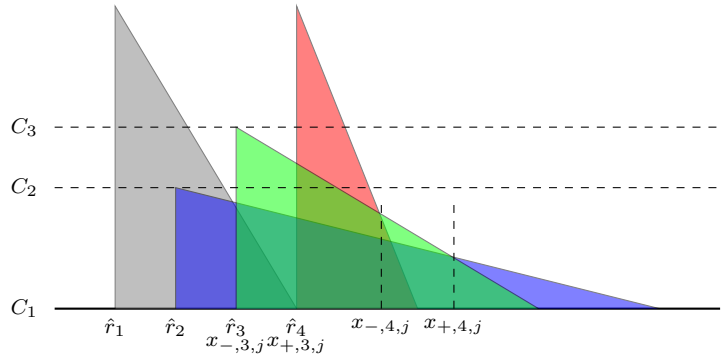


Fig. 6: Let j be the green job. The set $\Psi_j \coloneqq \{3, 4\}$ corresponds to the 2 execution intervals of j. The speeds at the border of the first execution interval, $s_{-,3,j}$ and $s_{+,3,j}$, are both equal to s_2. Similarly, the border speeds in the second execution interval, $s_{-,4,j}$ and $s_{+,4,j}$, are both equal to s_1. The value $q_{-,j} = 3$ refers to the first and $q_{+,j} = 4$ to the last execution interval of j. Finally, the indicator variables for j in the depicted example have the following values: $y_{+,j}(3) = y_{-,j}(3) = 1$ (borders at some release time), $y_{+,j}(4) = y_{-,j}(4) = 0$ (borders not at some release time), $\chi_j(3) = 1$ (not j's last execution interval), and $\chi_j(4) = 0$ (last execution interval of j).

right border of this execution interval. Let $s_{-,q,j}$ denote the speed at which j is running at the left endpoint corresponding to q and $s_{+,q,j}$ denote the speed j is running at the right endpoint. Let $q_{-,j}$ be the smallest and $q_{+,j}$ be the largest indices of Ψ_j (i.e., the indices of the first and last execution intervals of job j).

Let the indicator variable $y_{+,j}(q)$ denote whether $x_{+,q,j}$ occurs at a release point. Another indicator variable $y_{-,j}(q)$ denotes whether $x_{-,q,j}$ occurs at r_j. 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 $x_{+,q'-1,j} = x_{-,q',j}$, and either $\rho_j(q) + 1 \notin \Psi_j$ or $x_{+,\rho_j(q),j} \neq x_{-,\rho_j(q)+1,j}$.

Note that, 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., $\Psi_j(\tau)$, $x_{-,q,j}(\tau)$, etc.).

Events & Algorithm. 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 6 (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 $v \in (\tau_0, \tau_0 + \epsilon)$:

```
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 job completion}
7        (b) AffectionChange(S(τ), A_i(τ), [α'_1, α'_2, ..., α'_i])  {time to affection change}
8        (c) SpeedChange(S(τ), [α'_1, α'_2, ..., α'_i])              {time to speed change}
9        (d) RateChange(S(τ), i, [α'_1, α'_2, ..., α'_i])            {time to 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: Algorithm to compute an optimal schedule. Here, $\delta_{j,i}(\tau)$ describes how fast j's dual line is raised with respect to i's dual line (see Section 7.2).

- 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 job's interval changes. That is, there exists $j \in [i]$ and $q \in \Psi_j(\tau)$ such that either $s_{-,q,j}(u) \neq s_{-,q,j}(v)$ or $s_{+,q,j}(u) \neq s_{+,q,j}(v)$. 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 $x_{+,q,i}(u) < r_j$ and $x_{+,q,i}(v) = r_j$. 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.

7.2 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. 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

$$p_j = \sum_{q \in \Psi_j} \Bigg[s_{-,q,j} \left(\frac{\alpha_j - \check{C}(D_j^\tau(x_{-,q,j}))}{d_j} + r_j - x_{-,q,j} \right) + \sum_{a \colon s_{-,q,j} > s_a > s_{+,q,j}} s_a \left(\frac{\alpha_j - C_a}{d_j} + r_j - \left(\frac{\alpha_j - C_{a+1}}{d_j} + r_j \right) \right) + s_{+,q,j} \left(x_{+,q,j} - \left(\frac{\alpha_j - \hat{C}(D_j^\tau(x_{+,q,j}))}{d_j} + r_j \right) \right) \Bigg].$$

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

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

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:

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

We will prove in Lemma 9 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 $\delta_{a,b}$ over all edges (a,b) on the path from j to i. Similarly we can compute $\delta_{j,j'}$ for all $j, j' \in A_i$.

Lemma 7 *Intersection points on the upper envelope cannot move towards the right when τ is increased.*

Proof Since, by Lemma 6, parents in the affection tree are always of lower-density than their children, and since dual lines are monotonically decreasing, we have that $\delta_{\iota_j,j} \leq 1$. This implies the claim. □

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

Lemma 8 *Consider any job $j \in A_i$ whose dual line gets raised at a rate $\delta_{j,i}$.*

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

(b) For an interval $q \in \Psi_j$, if $\chi_j(q) = 1$, then $x'_{+,q,j} = 0$.
(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, $x'_{-,q_j,j} = x'_{+,q_{j'},j'} = -\frac{\alpha'_j - \alpha'_{j'}}{d_{j'} - d_j}$. 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 $x'_{+,q,i} = 0$ or $x'_{+,q,i} = 1/d_i$.

Proof

(a) Note that since $y_{-,j}(q) = 1$, this implies that $x_{-,q,r} = r_j$. Since by Lemma 7 intersection points can only move towards the left and by definition D^τ_j is defined in $[r_j, \infty)$ the statement follows.
(b) Set $t = x_{+,q,j}$ and let us consider two subcases. In the first case, assume that there exists an $\epsilon > 0$ such that j is run in $(t, t + \epsilon)$. Then, we must have that $t = x_{+,q,j} = r_{j'}$ for some $j' \neq j$, as otherwise q would not be maximal. This implies $x'_{+,q,j} = 0$.
In the second subcase, assume that there does not exist any $\epsilon > 0$ such that j is run in $(t, t + \epsilon)$. This implies there is some change in the upper envelope at t, which can happen only in the following three cases:
(i) The dual line crosses 0 at t. That is, $\alpha_j - d_j(t - r_j) = 0$.
(ii) The dual line crosses a dual line of smaller slope at t.
(iii) A release time causes a discontinuity on the upper envelope at t.
Note that (i) and (ii) can only happen at the last execution interval of a job, but since $\chi_j(q) = 1$, q is not the last interval in which j is run. In (iii), since $x_{+,q,j} = r_{j'}$ at a discontinuity, $x'_{+,q,j} = 0$ and the statement holds.
(c) Note that since (j, j') is an edge in the affection tree, by Observation 2 we have that $D^\tau_{j'}$ and D^τ_j must intersect on the (left) upper envelope. Since $D^\tau_{j'}(t) = \alpha_{j'} - d_{j'}(t - r_{j'})$ and $D^\tau_j(t) = \alpha_j - d_j(t - r_j)$, the dual lines for j and j' intersect at

$$t = \frac{\alpha_{j'} + d_{j'} \cdot r_{j'} - \alpha_j - d_j \cdot r_j}{d_{j'} - d_j}, \text{ and its derivative is } -\frac{\alpha'_j - \alpha'_{j'}}{d_{j'} - d_j}.$$

Since j is a parent of j', $x_{-,q_j,j} = x_{+,q_{j'},j'} = t$ and the result follows.
(d) Note that since job i has the lowest density of all jobs currently considered, its rightmost interval can only stop at a release time of a denser job, or at a point t such that $D^\tau_i = 0$. In the first case $x'_{+,q,i} = 0$. In the second case note that $D^\tau_i(t) = \alpha_i - d_i(t - r_i)$ intersects 0 at $t = \alpha_i/d_i + r_i$. Taking the derivative with respect to τ yields $x'_{+,q,i} = \alpha'_i/d_i = 1/d_i$, as desired. □

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 the corresponding τ values can be easily computed.

Lemma 9 *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 $j \in [i-1]$.*

Proof Clearly, by the definition of affection and construction of the affection tree, if $j' \notin A_i$, then by setting $\alpha'_{j'} = 0$ we have that $p'_{j'} = 0$.

For a parent and child $(\iota_j, j) \in A_i$, we set $p'_j = 0$ in Equation 4 and solve for ${}^{\alpha'_j}/_{\alpha'_{\iota_j}} = \delta_{j,\iota_j}$. Let $I_{q,j} = \{ q, \ldots, \rho_j(q) \}$ if $q \in \Psi_j$ and $\emptyset$ otherwise. We call $I_{q,j}$ a *maximal execution interval* of j if $I_{q-1,j} \cap I_{q,j} = \emptyset$. Let $M = \{ q \in \Psi_j \mid I_{q,j}$ is max. execution interval of $j \}$. We have that $\bigcup_{q \in M} I_{q,j} = \Psi_j$. Let $p'_{j,S}$ where $S \subseteq \Psi_j$ be the rate of change of p_j due to the rate of change of the endpoints of the intervals in S. If j is run at its release time, then $y_{-,j}(q_{-,j}) = 1$ and by, Observation 3, j cannot intersect any of its children at its release time, so by Lemma 8

$$p'_{j,I_{q_{-,j},j}} = (s_{-,q_{-,j},j} - s_{+,\rho_j(q_{-,j}),j}) \left(\frac{\alpha'_j}{d_j} \right) + s_{+,\rho_j(q_{-,j}),j} \left(x'_{+,\rho_j(q_{-,j}),j} \right).$$

For any other $q \in M$ (including $q_{-,j}$ if $y_{-,j}(q_{-,j}) = 0$), q must begin at the intersection point of j and one of its children. That is, there exists a unique $(j, j') \in A_i$ such that $q = q_{j,j'}$. Therefore, by Lemma 8

$$\begin{aligned} p'_{j,I_{q_{j,j'},j}} = \quad & (s_{-,q_{j,j'},j} - s_{+,\rho_j(q_{j,j'}),j}) \left(\frac{\alpha'_j}{d_j} \right) + s_{-,q_{j,j'},j} \left(\frac{\alpha'_j - \alpha'_{j'}}{d_{j'} - d_j} \right) \\ & + s_{+,\rho_j(q_{j,j'}),j} \left(x'_{+,\rho_j(q_{j,j'}),j} \right). \end{aligned}$$

For any $q \in M$, we have (Lemma 8) that $x'_{+,\rho_j(q),j} = 0$ if $\rho_j(q) \neq q_{+,j}$ and $x'_{+,q_{+,j},j} = -\frac{\alpha'_{\iota_j} - \alpha'_j}{d_j - d_{\iota_j}}$. The lemma follows by observing that $p'_j = \sum_{q \in M} p'_{j,I_{q,j}}$, the fact that j must intersect each of its children exactly once on the (left) upper envelope, and that for $(j, j') \in A_i$, we have that $\alpha_{j'}/\alpha_j = \delta_{j',j}$. □

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 10 *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 $\tilde{\tau} \in (\tau_0 - \epsilon, \tau_0)$, j' was not in the affection tree.

Proof It is straightforward to see that whenever (a), (b), or (c) occurs, an affection change event has to take place. Therefore we focus the rest of the proof on showing the other direction, i.e., any affection change event is always a consequence of one of the aforementioned cases.

By definition, any change in the affection tree is built from a sequence of edge additions and edge removals. We therefore will separately consider the cases where an edge is removed or added.

Case: An edge between j and j' is removed.

Let τ_0 be a time when an edge between j and j' is removed, and assume that j and j' had an intersection point t. Assume furthermore, without loss of generality, that j is a parent of j'. Therefore the affection $j \to j'$ ceases to exist at τ_0 (perhaps also $j' \to j$ if it existed). First note that at t, $D_j^{\tau_0}(t) = D_{j'}^{\tau_0}(t)$ must be on the upper envelope or left upper envelope at a discontinuity, otherwise there exists some $\epsilon > 0$ such that at time $\tau_0 - \epsilon$ their intersection was not on the upper envelope or the left upper envelope but the edge $j \to j'$ existed, contradicting Observation 2. We handle these two cases separately.

Subcase: $D_j^{\tau_0}(t) = D_{j'}^{\tau_0}(t)$ lies on upper envelope.

By Lemma 6, we know that it must be the case that $d_{j'} > d_j$. Furthermore, by Observation 3, since now it is the first time that $j \not\to j'$, either $r_j = t$ (which is covered in statement (a) of the lemma), or at least three jobs intersect at t. If this is the case, let j'' be the highest density job among the jobs that intersect at t. Note that j'' cannot be j (since $d_{j'} > d_j$) and it can also not be j' (since $j \not\to j'$).

By Observation 3, just before τ_0, j' does work to the left of the intersection point (between j and j') and j to the right. But at τ_0, j' cannot do any work directly to the left of t, because of j''. It follows that the interval of j' has disappeared, since to the left of t, j'' is run and to the right of t, j is run. This case is covered in the statement (b) of the lemma.

Subcase: $D_j^{\tau_0}(t) = D_{j'}^{\tau_0}(t)$ lies on left upper envelope at discontinuity t.

Note that since the intersection points do not move towards the right as τ increases (by Lemma 7), the intersection of j and j' was either at t or it was moving to the left towards t for all times during which $j \to j'$. However, since there is a discontinuity at t, there is some job j'' on the upper envelope that is not on the left upper envelope. If the intersection was at t then $j \to j'$ would not be possible. Therefore, there must exist some $\epsilon > 0$ such that at $\tau_0 - \epsilon$ the intersection of j and j' was below the curve of j''. This contradicts Observation 2. It follows that this subcase cannot occur.

Case: A new edge is added to the affection tree.

Let τ_0 be the time when a new edge is added to the affection tree. First note that, without loss of generality, at least one new edge is between two nodes j and j' with (a) j is in the affection tree immediately before τ_0, (b) j' is not in the affection tree immediately before τ_0, and (c) j becomes the

parent of j' at τ_0. Indeed, obviously at least one old node j of the affection tree must be involved as a parent in one of the new edges. Note that in the above cases "immediately before" may refer to either some interval $(\tau_0 - \epsilon, \tau_0)$, for an appropriate $\epsilon > 0$, or to the situation directly after some edge is removed at τ_0, and before adding the new edge.
If all new children j' of such old nodes were in the affection tree immediately before τ_0, there would also be some edge removal at τ_0 (as an additional edge would break the tree property, contradicting Lemma 6). This would reduce this case to the previous one, i.e., we consider edge removals at τ_0 before edge additions at τ_0.
From the above we get $d_j < d_{j'}$ (j being a parent of j'). Moreover, by Observation 2, at τ_0 the intersection point t of j and j' is on the (left) upper envelope and j' is run either to the left or right of t. Since j has a lower density, it must be run to the left of t. This is the case covered by statement (c) of the lemma. □

7.2.1 The Subroutines

There are four types of events that cause the algorithm to recalculate the rates at which it is raising the dual lines. In Lemma 10 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 7 and Lemma 8. Given the rates at which the algorithm raises the dual lines, we can easily calculate the time until the next event. This subsection gives a formal description of these subroutines and their correctness proofs.

Job Completion Event. Job completion events, which capture when the current job i is finished, are the easiest events to handle. As long as no other event occurs, the work of job i is processed at a constant rate p'_i, which can be computed by Equation (4) (using Lemma 8 to compute $x'_{l,q,i}$ and $x'_{+,q,i}$). With $p_i(\tau)$ denoting the work of i processed at the current time τ, we define

$$\Delta\tau := \frac{p_i - p_i(\tau)}{\sum_{q \in \Psi_i(\tau)} \left[s_{-,q,i} \left(\frac{\alpha'_i(\tau)}{d_i} - x'_{-,q,i}(\tau) \right) + s_{+,q,i} \left(x'_{+,q,i}(\tau) - \frac{\alpha'_i(\tau)}{d_i} \right) \right]}. \quad (6)$$

With the above discussion, we immediately get the following lemma.

Lemma 11 *Assume the next event is a job completion event. Then this event occurs at $\tau + \Delta\tau$, with $\Delta\tau$ as computed by our job completion subroutine via Equation 6.*

Simple Rate Change Event. Simple rate change events are similarly easy to compute. These occur when the right side of i's last execution interval reaches the release time of some job. By Lemma 8(d), this happens only when the

rate at which this interval border changes jumps from $1/d_i$ to 0. Thus, the corresponding time is computed as

$$\Delta\tau \coloneqq (\hat{r}_{q_{+,i}(\tau)+1}(\tau) - x_{+,q_{+,i}(\tau),i}(\tau)) \cdot d_i. \tag{7}$$

This yields the following lemma.

Lemma 12 *Assume the next event is a simple rate change event. Then this event occurs at $\tau + \Delta\tau$, with $\Delta\tau$ as computed by our simple rate change subroutine via Equation 7.*

Speed Change Event. While the basic idea is the same as for the previous events, computations for speed change events are a bit more tedious. For each execution interval of each job, we have to check when the job's dual function value at the interval borders crosses a speed threshold. See Algorithm 2 for the actual computations. We prove its correctness in Lemma 13.

Lemma 13 *Assume the next event is a speed change event. Then this event occurs at $\tau + \Delta\tau$, with $\Delta\tau$ as computed by our speed change subroutine shown in Algorithm 2.*

Proof Consider the innermost "for loop" of Algorithm 2. It computes the time until the left or right border of the q-th execution interval of j reaches a speed threshold, assuming that no other event occurs before. To see this, note that if the interval has length 0 and not increasing in size (the *else* branch), no work will be done in this interval until the next event. Thus, in this case q cannot cause the next event. Otherwise (the *if* branch), we know that $x'_{-,q,j}(\tau) \leq 0$ and $\alpha'_j(\tau) \geq 0$, and these rates remain constant until the next event. Thus, the dual function value $D_j^{\tau'}(x_{-,q,j}(\tau'))$ at the left border of q is always non-decreasing for $\tau' > \tau$ (until the next event). Thus, the speed $s_{-,q,j}$ at this interval border remains constant until $\tau' > \tau$ with $D_j^{\tau'}(x_{-,q,j}(\tau')) = \hat{C}(D_j^{\tau}(x_{-,q,j}(\tau)))$. With $\Delta\tau_{-,q,j} = \tau' - \tau$, we can write

$$\begin{aligned} D_j^{\tau'}(x_{-,q,j}(\tau')) &= \alpha_j(\tau') - d_j(x_{-,q,j}(\tau') - r_j) \\ &= \alpha_j(\tau) + \Delta\tau_{-,q,j}\alpha'_j(\tau) - d_j\big(x_{-,q,j}(\tau) + \Delta\tau_{-,q,j}x'_{-,q,j}(\tau) - r_j\big) \\ &= D_j^{\tau}(x_{-,q,j}(\tau)) + \Delta\tau_{-,q,j}\big(\alpha'_j(\tau) - d_j x'_{-,q,j}(\tau)\big). \end{aligned}$$

If we set this equal to $\hat{C}(D_j^{\tau}(x_{-,q,j}(\tau)))$ and solve for $\Delta\tau_{-,q,j}$, we get exactly the value computed by Algorithm 2. Analogously, we see that $\Delta\tau_{+,q,j}$ are computed correctly. Finally, the algorithm set $\Delta\tau$ to the first of all these computed events. □

for each job $j \in [i]$:
  for each $q \in \Psi_j(\tau)$:
    if $x_{+,q,j} \neq x_{-,q,j}$ or $x'_{+,q,j} - x'_{-,q,j} > 0$:
      $\Delta\tau_{-,q,j} = \frac{\hat{C}(D_j^\tau(x_{-,q,j}(\tau))) - D_j^\tau(x_{-,q,j}(\tau))}{-x'_{-,q,j}(\tau)d_j + \alpha'_j(\tau)}$
      $\Delta\tau_{+,q,j} = \frac{\hat{C}(D_j^\tau(x_{+,q,j}(\tau))) - D_j^\tau(x_{+,q,j}(\tau))}{-x'_{+,q,j}(\tau)d_j + \alpha'_j(\tau)}$
    else:
      $\Delta\tau_{-,q,j} = \Delta\tau_{+,q,j} = \infty$
$\Delta\tau = \min_{j\in[i],q\in\Psi_j(\tau)}(\min(\Delta\tau_{-,q,j}, \Delta\tau_{+,q,j}))$
return $\Delta\tau$

Algorithm 2: SpeedChange($S(\tau), [\alpha'_1, \alpha'_2, \ldots, \alpha'_i]$).

Affection Change Event. The last event type is the most involved. However, Lemma 10 gives the exact conditions for when and why an affection change event can occur. More precisely, Lemma 10(a) and 10(b) correspond to edge removals in the affection tree, while Lemma 10(c) corresponds to an addition of an edge. The computations of all such possible events is formalized in Algorithm 3. Lemma 14 states and proves its correctness.

Lemma 14 *Assume the next event is an affection change event. Then this event occurs at $\tau + \Delta\tau$, with $\Delta\tau$ as computed by our affection change subroutine shown in Algorithm 3.*

Proof By Lemma 10, an edge (and hence a job) is removed from the affection tree only when a nonzero interval becomes 0. Thus for any job j in the tree and nonzero interval q of it, the rate of change of the size of that interval is $v = x'_{+,q,j}(\tau) - x'_{-,q,j}(\tau)$, which is negative if the size of the interval is decreasing. The size of the interval is $x_{+,q,j}(\tau) - x_{-,q,j}(\tau)$, thus at rate v it will become zero at $\Delta\tau_{j,q}$.

By Lemma 10, an edge (and hence job) is added to the affection tree only when a job j not in the affection tree intersects a job a in the affection tree at the right endpoint of a nonzero interval of j. Since j is not in the affection tree, its endpoints do not change as τ increases. For some interval q, the distance between D_a^τ and D_j^τ at the right endpoint of q is $D_j^\tau(x_{+,q,j}(\tau)) - D_a^\tau(x_{+,q,j}(\tau))$, and D_a^τ increases at a rate of α'_a. □

7.2.2 Completing the Correctness Proof

We are now ready to prove the correctness of the algorithm. We handle termination in Theorem 8, where we prove a polynomial running time for our algorithm.

Theorem 7 *Assuming that Algorithm 1 terminates, it computes an optimal schedule.*

$\{$ *all* $\tau_{j,q}$ *and* $\tau_{j,q,a}$ *initialized with* $\infty\}$
for $j \in A_i(\tau)$: $\{$*calculate affection tree removals*$\}$
  for $q \in \Psi_j(\tau)$:
    if $x_{-,q,j}(\tau) \neq x_{+,q,j}(\tau)$ and $x'_{+,q,j}(\tau) - x'_{-,q,j}(\tau) < 0$: $\{q$ *shrinks*$\}$
      $\Delta\tau_{j,q} = \frac{x_{+,q,j}(\tau) - x_{-,q,j}(\tau)}{x'_{-,q,j}(\tau) - x'_{+,q,j}(\tau)}$
$\Delta\tau_1 = \min_{j \in A_i(\tau), q \in \Psi_j(\tau)}(\Delta\tau_{j,q})$

for $j \in [i] \setminus A_i(\tau)$: $\{$*calculate affection tree additions*$\}$
  for $q \in \Psi_j(\tau)$:
    for $a \in A_i(\tau)$:
      if $x_{-,q,j} \neq x_{+,q,j}$ and $r_a < x_{+,q,j}$:
        $\Delta\tau_{j,q,a} = \frac{D_j^\tau(x_{+,q,j}(\tau)) - D_a^\tau(x_{+,q,j}(\tau))}{\alpha'_a}$

$\Delta\tau_2 = \min_{j \in [i] \setminus A_i(\tau), q \in \Psi_j(\tau), a \in A_i(\tau)}(\Delta\tau_{j,q,a})$
$\Delta\tau = \min(\Delta\tau_1, \Delta\tau_2)$
return $\Delta\tau$

Algorithm 3: AffectionChange$(S(\tau), A_i(\tau), [\alpha'_1, \alpha'_2, \ldots, \alpha'_i])$.

Proof The algorithm outputs a line schedule S, so by Theorem 1, 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. The lemmas from Section 7.2.1 state that all these events are recognized (which causes the algorithm to recalculate the rates). By Lemma 9 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 recognizes by Lemma 11. 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. □

7.3 A Note on Density Uniqueness

For two jobs i and j with different densities $d_i \neq d_j$, the dual lines $D_i^{a_i}$ and $D_j^{a_j}$ intersect in at most one point t^*. Therefore the only time they can both be on the upper envelope (or left upper envelope) is t^*. However, if $d_i = d_j$ and $D_i^{a_i}$ and $D_j^{a_j}$ intersect once, then they intersect at every time after both i and j have been released, and thus both may be on the upper envelope for entire intervals. We resolve any ambiguity by imposing the rule that if $d_i = d_j$, $r_i < r_j$, and $D_i^{a_i}$ and $D_j^{a_j}$ intersect, then i must complete all its work before j completes any work (if $r_i = r_j$, we arbitrarily pick one to complete first).

Let $J = \{ j_1, \ldots, j_z \}$ be the largest set of jobs that all have some density d_j and intersect, ordered by release time. We say that the intersection between $D_{j_i}^{a_{j_i}}$ and $D_{j_{i+1}}^{a_{j_{i+1}}}$ occurs at the time at which j_i has completed p_{j_i} work, if such a time exists. For any other pair of jobs $j_i, j_\ell \in J$ such that $|i - \ell| > 1$, we say that $D_{j_i}^{a_{j_i}}$ and $D_{j_\ell}^{a_{j_\ell}}$ do not intersect. Thus for the purpose of having a well-defined upper envelope, we consider a job $j_\ell \in J$ to be on the upper envelope at t only when $D_{j_\ell}^{a_\ell}$ is the highest curve at t, every other job $j_i \in J$ with $i \in [\ell - 1]$ has completed by t, and either j_ℓ has not completed by t or $\ell = z$.

We now discuss how the algorithm must be modified when there exist jobs with equal densities. Fix an iteration i of the algorithm and a τ. If $(j, \iota_j) \in A_i$ with $d_j = d_{\iota_j}$, we set $\delta_{j,\iota_j} = 1$. All that remains is to show how the rate of change of the intersection point between j and ι_j (as defined above) can be computed, i.e., $x'_{-,q_{j,\iota_j},j}$, since now Lemma 8(c) calculates a value that is not well-defined. This is calculated such that $p'_j = 0$. In other words, by Equation 4 and Lemma 8(b) we have that

$$x'_{-,q_{j,\iota_j},j} = -\frac{1}{s_{+,q_{+,j},j}} \left(\sum_{q \in \Psi_j} (s_{-,q,j} - s_{+,q,j}) \frac{\alpha'_j}{d_j} - s_{-,q,j} x'_{-,q,j} \right). \tag{8}$$

Thus if we know the rates of change of the intersection points for j and its children, we can calculate the rate of change of j's intersection with ι_j. If $(\iota_j, u) \in A_i$ and $d_{\iota_j} > d_u$, then $\delta_{\iota_j,u}$ will be calculated differently than shown in Equation 5, but the necessary change is a simple replacement of the term $(1 - \delta_{j,\iota_j})/(d_j - d_{\iota_j})$ with the term $x'_{-,q_{j,\iota_j},j}/\alpha'_{\iota_j}$.

8 The Running Time

The purpose of this section is to analyze the running time of Algorithm 1 by proving the following theorem.

Theorem 8 *Algorithm 1 takes $O\left(n^4 k\right)$ time.*

We use the following approach in order to prove Theorem 8. Details follow below.

- We give upper bounds on the total number of events that can occur in Lemma 16. This is relatively straightforward for job completion, simple rate change, and speed change events, which can occur $O(n)$, $O\left(n^2\right)$, and $O\left(n^2 k\right)$ 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 $O\left(n^2\right)$ as well.
- We show in Lemma 17 that the next event can always be calculated in $O\left(n^2\right)$ time.

– We show in Lemma 18 that the affection tree can be updated in $O(n)$ time after each affection change event.

The above results imply that our algorithm has a running time of $O(n^4k)$, and therefore Theorem 8 follows.

We start with an auxiliary lemma that will be useful for bounding the number of affection change events in Lemma 16.

Lemma 15 *Consider some time τ_0 where an edge (j, j') is removed from the affection tree. Then, there exists some edge (u, v) that is also being removed at τ_0 such that (u, v) will not be present for all remaining iterations of the algorithm.*

Proof First note that by the definition of the affection tree, it must be that the affection $j \to j'$ is being removed. Since j is a parent of j', by Lemma 6 we have $d_j < d_{j'}$. Also, by Lemma 10, this edge can be removed because either the intersection between j and j' becomes equal to r_j or two intersection points become equal. We handle these cases separately.

In the first case we show that the affection edge $j \to j'$ cannot be present again. To do this, we show that the invariant of j' not being processed on the (left) upper envelope to the right of r_j is always maintained. This implies that the edge $j \to j'$ is never present again. It is clearly true for τ_0 (say, in iteration i). Assume that for some iteration $i' \geq i$ this invariant is true. If j' is not being raised the invariant will remain true since curves can only be raised and not lowered. If j' is being raised, since it is not the lowest density job it must intersect some lower density job j'' (its parent) that is also being raised. Further, since the invariant is true to this point, the intersection is not to the right of r_j. However, while j' is being raised, by Lemma 7 the intersection between j' and j'' moves only to the left. Since $d_{j''} < d_{j'}$, j' will not be on the upper envelope or left upper envelope to the right of this intersection and the result follows.

In the second case, assume that the intersection between j_1 and j_2 becomes equal to the intersection between j_2 and j_3 and assume without loss of generality that $d_{j_1} < d_{j_2} < d_{j_3}$. This implies the edges (j_1, j_2) and (j_2, j_3) will be removed. We show that the edge (j_2, j_3) will not be present again. First note that $r_{j_2} < r_{j_3}$ since otherwise j_2 would not be processed anywhere, contradicting that the rates at which we raise curves are work-preserving. Similar to the previous argument, we show that j_2 will not be processed on the upper envelope or left upper envelope to the right of r_{j_3} again. This is clearly true at τ_0 (say, in iteration i). Assume for some iteration $i' \geq i$ this invariant is true. Again, if j_2 is not being raised the invariant remains true. If j_2 is being raised, it must intersect a lower density job (its parent) to the left of r_{j_3}. Since this intersection point will move only to the left the result follows. □

Lemma 16 *The total number of events throughout the execution of the algorithm is $O(n^2k)$.*

Proof To show this we show that the number of events is $O(n^2k)$ for each single type.

Job completion event: Note that a job completion event occurs exactly once per iteration: After a job completion event occurs on iteration i, we have that $p_i(\tau) = p_i$ and in turn the algorithm moves to the next iteration. Therefore, the total number of job completion events that occur is exactly n.

Simple rate change event: Consider iteration i. A simple rate change event can occur, if and only if, the last interval of job i changes from ending before the release time of some job to ending at the release time of the job. Note that since dual lines are never lowered during the execution of the algorithm this can occur at most once for each release time and therefore at most n times. As we have n iterations the total number of simple rate change events is $O\left(n^2\right)$

Affection change event: These events happen when an edge is either added or removed in the tree. Note that the number of edge additions is bounded by 2 times the number of edge removals, so it suffices to bound the number of removals. By Lemma 15, for each (possibly temporarily) removed edge at least one edge is removed permanently. Thus, the total number of such events is $O\left(n^2\right)$.

Speed change event: A speed change event occurs when the right or left endpoint of an interval for a job j crosses a speed threshold. We first show that each speed threshold can be crossed at most twice per interval, once by the left and once by the right endpoint of the interval. Consider an interval I for job j. If I is never removed, then each endpoint of I can only cross each speed threshold at most once, since by Lemma 7 the endpoints of intervals only move towards left, and furthermore no dual line is ever lowered during the execution of the algorithm. Else, if I is removed then the left and right endpoints of I coincide on the upper envelope at some point t (by Lemma 10) and there must be some other job j' of lower-density whose dual-line also intersects at t. However, by Lemma 7, the left endpoint of j' (in case more than two dual-lines intersect at t let j' be the job corresponding to such a dual line of lowest-density) will only move to the left while this interval is not present. Therefore, even if interval I does reappear, the left and right endpoints will not be at lower speeds.

Finally, since each job has at most n intervals and each such interval can cause at most $2k$ speed change events, the total number of speed change events is $O\left(n^2k\right)$. □

Lemma 17 *Calculating the next event takes $O\left(n^2\right)$ time.*

Proof We start by noting that the total number of different intervals during the execution of the algorithm is $O(n)$. This follows by the fact that a new interval can only be introduced when a new job gets released, or a job completes its execution.

To calculate the next event, we look at each event type and calculate how far in the future the next event of this type will occur. Then we just choose the event of the type that will happen sooner. Therefore it suffices to give bounds

on the time required to calculate the next event of each type (cf. subroutines in Section 7.2.1).

Affection change event: An affection change event has to be either a removal or an addition (see Lemma 14). If it is a removal, then by the observation on the number of intervals, it can be calculated in $O(n)$ time. On the other hand if the next affection change event is an addition, then again by the above observation on the number of intervals $O(n^2)$-time is required.

Speed change event: For any fixed interval the next speed change event can be calculated in constant time. Therefore, by the observation on the number of intervals, we have that the next such event over all jobs can be computed in time $O(n)$,

Simple rate change event: $O(n)$-time is sufficient in order to identify $q_{+,i}$ and $\hat{r}_{q_{+,i}+1}$, and therefore also to calculate this type of event as well.

Job completion event: We have to calculate $y_{+,j}, y_{-,j}$ for each of the $O(n)$ intervals, identify i' and calculate $\delta_{i,i'}$. Therefore we can calculate the next job completion event in time $O(n)$.

Combining the above, we can calculate the next event in $O(n^2)$ time. □

Lemma 18 *Updating the affection tree takes $O(n)$ time.*

Proof A simple way to update the affection tree is by recomputing it from scratch at each update. By Lemma 1, jobs in the tree always have a higher density than their parents. Further, by Observations 3 and 4, if a job j is on the upper envelope (or left upper envelope) at some time t and has release time before t, and $j' \neq j$ is the highest-density job on the upper envelope (left upper envelope) at time t, then $j \to j'$, and for any other job $j'' \neq j'$ of higher density than j on the upper envelope (left upper envelope), $j \not\to j''$. Therefore, for any job j, its children in the affection tree are those highest-density jobs that intersect it on the left endpoint of any of its intervals that begin after j's release. Thus, to compute the affection tree, we can iterate through each interval I of job i that begins after its release, add as i's children the highest density jobs that intersect it at I's left endpoint, and recursively do the same for i's children. By the observation that there are at most $2n$ intervals, this takes at most $O(n)$ time. □

9 Conclusion

We introduced an algorithm to find an optimal energy and fractional weighted flow trade-off schedule. Our approach is new in that it uses a geometric interpretation of optimality conditions obtained via a primal-dual formulation of the problem. This geometric interpretation allows for a very intuitive algorithm, and we believe that this approach might help to solve other, related problems. The most interesting open question left is to consider our problem for arbitrary, non-discrete speeds. In particular, it would be interesting whether a similar geometric interpretation can be used to guide an optimization algorithm for arbitrary speeds.

Compliance with Ethical Standards

Funding: The different authors (and thereby this work) were supported by: A fellowship within the Postdoc-Programme of the German Academic Exchange Service (A. Antoniadis); the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1247842 (N. Barcelo); the NSF Graduate Research Fellowship DGE-1038321 (M. Consuegra); 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 (P. Kling); by NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award (K. Pruhs); by a fellowship of Fondazione Ing. Aldo Gini, University of Padova, Italy (M. Scquizzato).

Conflict of Interest: The authors declare that they have no conflict of interest.

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. Antonios Antoniadis, Neal Barcelo, Mario Consuegra, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. Efficient computation of optimal energy and fractional weighted flow trade-off schedules. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 63–74, 2014.
5. 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.
6. Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.
7. Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2), 2013.
8. 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.
9. Neal Barcelo, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. On the complexity of speed scaling. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 75–89, 2015.

10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.