On the Complexity of Speed Scaling

Neal Barcelo[1⋆], Peter Kling[2⋆⋆], Michael Nugent[1], Kirk Pruhs[1⋆⋆⋆], and Michele Scquizzato[3†]

[1] Department of Computer Science, University of Pittsburgh, Pittsburgh, USA
[2] School of Computing Science, Simon Fraser University, Burnaby, Canada
[3] Department of Computer Science, University of Houston, Houston, USA

Abstract. The most commonly studied energy management technique is speed scaling, which involves operating the processor in a slow, energy-efficient mode at non-critical times, and in a fast, energy-inefficient mode at critical times. The natural resulting optimization problems involve scheduling jobs on a speed-scalable processor and have conflicting dual objectives of minimizing energy usage and minimizing waiting times. One can formulate many different optimization problems depending on how one models the processor (e.g., whether allowed speeds are discrete or continuous, and 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). There are a handful of papers in the algorithmic literature that each give an efficient algorithm for a particular formulation. In contrast, the goal of this paper is to look at a reasonably full landscape of all the possible formulations. We give several general reductions which, in some sense, reduce the number of problems that are distinct in a complexity theoretic sense. We show that some of the problems, for which there are efficient algorithms for a fixed speed processor, turn out to be NP-hard. We give efficient algorithms for some of the other problems. Finally, we identify those problems that appear to not be resolvable by standard techniques or by the techniques that we develop in this paper for the other problems.

1 Introduction

The most commonly studied energy management technique is speed scaling. It involves operating the processor in a slow, energy-efficient mode at non-critical times, and in a fast, energy-inefficient mode at critical times. The natural resulting optimization problems involve scheduling jobs on such a processor and have conflicting dual objectives of minimizing both energy usage and waiting times. This leads to many different optimization problems, depending on how one models the processor, the performance objective, and how one handles the dual objectives. There are several papers in algorithmic literature that give an efficient algorithm for a particular formulation. In contrast, we strive to look at a reasonably full landscape of all the possible formulations. We give several general reductions which reduce the number of problems that are distinct in a complexity theoretic sense. We show that some of the problems, for which there are efficient algorithms for a fixed speed processor, turn

⋆ This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1247842.
⋆⋆ Supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD). Work done while at the University of Pittsburgh.
⋆⋆⋆ Supported by NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award.
† Work done while at the University of Pittsburgh.

out to be NP-hard. We give efficient algorithms for some of the other problems. Finally, we identify those problems that appear to not be resolvable by standard techniques or by the techniques that we develop in this paper for the other problems.

Models. We now describe the different models that have been considered in the literature. Given the multitude of problems, we use a succinct representation, which we introduce in parenthesis.

– *Energy Budget (B) vs. Flow plus Energy (FE):* In an energy budget problem, the energy objective is turned into a constraint that the total energy used is at most some budget B. This setting is most appropriate when the energy is limited to some finite supply, such as the battery of a laptop. In a flow plus energy problem, the performance and energy objectives are linearly combined into a single objective. We use a constant coefficient β for the energy objective that, intuitively, represents the desired trade-off between the value of energy and the value of performance.

– *Integral Flow (I) vs. Fractional Flow (F):* In an integral flow problem, the objective is total (weighted) flow/waiting time of the jobs. In a fractional flow problem, the objective is total (weighted) flow of the work (job parts). If there is no benefit in partially completing a job, then integral flow is the right performance metric. Otherwise, fractional flow may be more appropriate.

– *Continuous Speeds (C) vs. Discrete Speeds (D):* In the discrete speed setting, the processor has a finite collection of allowable speeds and corresponding powers at which it may run. In the continuous speed setting, the allowable speeds are the nonnegative real numbers. While the discrete speed model is more realistic, it is often mathematically convenient to assume continuous speeds.

– *Weighted (W) vs. Unweighted (U):* In the unweighted setting, each job is of equal importance and is weighted equally in the performance objective. However, the raison d'être for power heterogeneous technologies, such as speed-scalable processors, is ubiquity of heterogeneity in the jobs. In the weighted case, the flow of jobs/work is weighted by their importance.

– *Arbitrary Size (A) vs. Unit Size (U):* In the unit size setting, each job has the same amount of work. Similar sized jobs occur in many information technology settings (e.g., for name servers). In the arbitrary size setting, the jobs may have different sizes.

– *Power Function:* In the continuous speed setting, one needs to model how a speed s maps to power. There are two common assumptions: Most commonly one assumes $P(s) = s^\alpha$ for a constant α, slightly generalizing the well-known cube-root rule that speed is approximately the cube-root of dynamic power. The second common assumption is that $P(s)$ is a general "nice" convex function. Intuitively, the complexity of speed scaling should not come from the power function's complexity. Foreshadowing slightly, our results support this intuition.

Previous Results. We now summarize the known complexity theoretic and offline algorithmic results using the succinct notation we just introduced. The format of our description is essentially a 5-tuple of the form $\star$-$\star\star\star\star$. The first entry captures the objective (**B**udget or **F**low plus **E**nergy). The remaining entries are **I**ntegral or **F**ractional flow, **C**ontinuous or **D**iscrete speed, **W**eighted or **U**nweighted, and **A**rbitrary or **U**nit size. A $\star$ represents a "don't care" entry. See Table 1 for an overview that puts all these results into the context of the full range of possible problems.

– `B-ICUU`: [17] gave a polynomial-time homotopic optimization algorithm for the problem of minimizing integral flow (I) with continuous speeds (C) subject to an energy budget (B)

for unweighted jobs (U) of unit size (U). They also used the assumption that $P(s) = s^\alpha$. The key insights were that jobs should be scheduled in FIFO order and that the KKT conditions for the natural convex program can be used to guide the homotopic search.

– `FE-ICUU`: [2] gave a polynomial-time dynamic programming algorithm for the problem of minimizing integral flow (I) with continuous speeds (C) for unweighted jobs (U) of unit size (U) and the objective of flow plus β energy (FE). Again, they were under the assumption that the power function was $P(s) = s^\alpha$, and again the key insight was that jobs should be scheduled in FIFO order.

– `FE-FCWA`: [10] gave a (not necessarily polynomial-time) homotopic optimization algorithm for the problem of minimizing fractional flow (F) with continuous speeds (C) for weighted jobs (W) of arbitrary size (A) and the objective of flow plus β energy (FE). The algorithm guides its search via the KKT conditions of the natural convex program.

– `FE-FDWA`: [4] gave a polynomial-time algorithm for the problem of minimizing fractional flow (F) with discrete speeds (D) for weighted jobs (W) of arbitrary size (A) and the objective of flow plus β energy (FE). The algorithm constructed an optimal schedule job by job, using the duality conditions of the natural linear program to find a new optimal schedule when a new job is added.

– `*-I*WA`: NP-hardness for integral flow (I) and weighted jobs (W) of arbitrary size (A) for the objective of flow plus β energy (FE) follows from NP-hardness of weighted integral flow for fixed speed processors [14] (via a suitable power function). The same holds for weighted integral flow (I) for weighted jobs (W) of arbitrary size (A) subject to a budget (B).

Our Results. The goal in this paper is to more fully map out the landscape of complexity and algorithmic results for the range of problems reflected in Table 1. In particular, for each setting our aim is to either give a combinatorial algorithm (i.e., without the use of a convex program) or show it is NP-hard. Let us summarize our results:

Hardness Results:

– *`B-IDUA` is NP-hard:* The reduction is from the subset sum problem. The basic idea is to associate several high density and low density jobs with each number in the subset sum instance, and show that for certain parameter settings, there are only two possible choices for this set of jobs, with the difference in energy consumption being this number.

– *`B-IDWU` is NP-hard:* A reduction similar to `B-IDUA`, but more technical.

These results are a bit surprising as, unlike the previous NP-hardness results for speed scaling in Table 1, these problems are either open or can be solved in polynomial time on a fixed speed processor.

Polynomial Time Algorithms:

– *`FE-ICUU` is in P:* We extend [2] to general power functions. This follows by noticing that a certain set of equations can be solved for general "nice" power functions.

– *`FE-IDUU` is in P:* The algorithm utilizes the structure of the unit size unweighted case. Here, discrete speeds allow for a much simpler algorithm than for `FE-ICUU`.

– *`FE-FCWA` is in P:* We generalize [4]'s algorithm to continuous speeds. The main hurdle is a more complicated equation system at certain points in the algorithm.

Equivalence Reductions:

– *Reduction from `B-FC**` to `FE-FC**`:* We reduce any energy budget problem with fractional flow and continuous speeds to the corresponding flow plus β energy problem using binary search.

- *Reduction from* `B-ICUU` *to* `FE-ICUU`*:* The difficulty here stems from the fact that there may be multiple optimal flow plus energy schedules for a β (so binary search over β does not suffice).
- *Reduction from* `*-*D**` *to* `*-*C**`*:* We give a reduction from any discrete speed problem to the corresponding continuous speed problem.

While not explicitly needed for our main results, we also provide the other directions for the first two reductions, in order to improve the understanding of structural similarities.

Table 1 summarizes our results and sets them into context of previous work. Note that for some of the problems shown to be solvable in polynomial time we give a direct algorithm, while others follow by one of our reductions. For problems that are solvable by reduction to linear/convex programming, our algorithms are faster and simpler than general linear/convex programming algorithms. The key takeaways from this more holistic view of the complexity of speed scaling problems are:

Certain parameters are sufficient for determining complexity:

- *Fractional Flow:* Looking at the first two rows, we see that any problem involving fractional flow can be solved in polynomial time. This generally follows from the ability to write the problem as a convex program, although we can give simpler and more efficient algorithms.
- *Integral Flow:* For integral flow there is a more fine grained distinction:
  - Weighted & Arbitrary Size: Everything with these settings is NP-hard. Given the NP-hardness of weighted flow for a fixed speed processor, this is not surprising.
  - Unweighted & Unit Size: Everything with these settings can be solved in polynomial time largely because FIFO is the optimal ordering of the jobs.
  - Unweighted & Arbitrary or Weighted & Unit size: These seem to be the most interesting settings (w.r.t. complexity). We show their hardness for a budget, but flow plus energy remains open.

Complexity of budget problem vs. flow plus energy: For every setting for which the complexity of each is known, the complexities (in terms of membership in P or NP-hardness) match. This might be seen as circumstantial evidence that the resolution to the remaining open complexity questions is that they are NP-hard. If these open problems do indeed have polynomial algorithms, it will require new insights as there are clear barriers to applying known techniques to these problems.

Other Related Work. There is a fair number of papers that study approximately computing optimal trade-off schedules, both offline and online. [16] also gives PTAS's for minimizing total flow without release times subject to an energy budget in both the continuous and discrete speed settings. [2, 3, 5–7, 11, 12, 15] consider online algorithms for optimal total flow and energy, [5, 7, 12] considers online algorithms for fractional flow and energy. In particular, [7] show that there are $O(1)$-competitive algorithms for all of the flow plus β energy problems that we consider (with arbitrary power functions). For a survey on energy-efficient algorithms, see [1]. For a fixed speed processor, all the fractional problems can be solved by running the job with highest density (=weight/size). Turning to integral flow, if all jobs are unit size, then always running the job of highest weight is optimal. The complexity of the problem if all jobs have the same (not unit) size is open [8, 9]. The complexity of `FE-I*WU` seems at least as hard (but perhaps not much harder) than this problem. If all jobs have unit weight, then Shortest Remaining Processing Time is optimal for total flow.

		Unweighted Jobs		**W**eighted Jobs	
		Unit Sizes	**A**rbitrary Sizes	**U**nit Sizes	**A**rbitrary Sizes
Fractional Flow	**D**iscrete Speeds	P [4] / P [⋆]	P [4] / P [⋆]	P [4] / P [⋆]	P [4] / P [⋆]
	Continuous Speeds	P [⋆] / P [⋆]	P [⋆] / P [⋆]	P [⋆] / P [⋆]	P [⋆] / P [⋆]
Integral Flow	**D**iscrete Speeds	P [⋆] / P [⋆]	? / NP-hard [⋆]	? / NP-hard [⋆]	NP-hard [14] / NP-hard [16]
	Continuous Speeds	P [2][⋆] / P [17][⋆]	? / NP-hard [⋆]	? / NP-hard [⋆]	NP-hard [14] / NP-hard [16]

Table 1. Summary of known and new results. Each cell's upper-half refers to the flow $+\,\beta\cdot$energy objective and the lower-half refers to flow minimization subject to an energy constraint. Results of this paper are indicated by [⋆], and $\equiv$ indicates that two problems are computationally equivalent.

Outline of the Paper. Section 2 provides basic definitions. In Section 3 we show that `B-IDWU` and `B-IDUA` are NP-hard. In Section 4 we give several polynomial time algorithms. Finally, in Section 5, we give the reductions between budget and flow plus β energy problems. Due to space constraints, omitted proofs are left to the full version of the paper.

2 Model & Notation

We consider n jobs $J=\{1,2,...,n\}$ to be processed on a single, speed-scalable processor. In the *continuous* setting, the processor's energy consumption is modeled by a power function $P:\mathbb{R}_{\geq 0}\to\mathbb{R}_{\geq 0}$ mapping a speed s to a power $P(s)$. We require P to be continuous, convex, and non-decreasing. Other than that, we merely assume P to be "nice" in the sense that we can solve basic equations involving the power function and, in particular, its derivative and inverse. In the *discrete* setting, the processor features only k distinct speeds $0<s_1<s_2<\cdots<s_k$, where a speed s_i consumes energy at the rate $P_i\geq 0$. Even in the discrete case, we will often use $P(s)$ to refer to the power consumption when "running at a speed $s\in(s_i,s_{i+1})$" in between the discrete speeds. This is to be understood as interpolating the speed $s=s_i+\gamma(s_{i+1}-s_i)$ (running for a γ fraction at speed s_{i+1} and a $1-\gamma$ fraction at speed s_i), yielding an equivalent discrete schedule. Each job $j\in J$ has a release time r_j, a processing volume p_j, and a weight w_j. The density of j is w_j/p_j. For each time t, a schedule S must decide which job to process at what speed. Preemption is allowed, so that a job may be suspended and resumed later on. We model a schedule S by a speed function $\mathcal{V}:\mathbb{R}_{\geq 0}\to\mathbb{R}_{\geq 0}$ and a scheduling policy $\mathcal{J}:\mathbb{R}_{\geq 0}\to J$. Here, $\mathcal{V}(t)$ denotes the speed at time t, and $\mathcal{J}(t)$ the job that is scheduled at time t. Jobs can be processed only after they have been released. For job j let $I_j=\mathcal{J}^{-1}(j)\cap[r_j,\infty)$ be the set of times during which it is processed. A feasible schedule must finish the work of all jobs. That is, the inequality $\int_{I_j}\mathcal{V}(t)dt\geq p_j$ must hold for all jobs j.

We measure the quality of a given schedule S by means of its energy consumption and its fractional or integral flow. The energy consumption of a job j is $E_j=\int_{I_j}P(\mathcal{V}(t))dt$, and

the energy consumption of schedule S is $\sum_{j\in J} E_j$. The *integral flow* $F_j = w_j(C_j - r_j)$ of a job j is the weighted difference between its completion time C_j and release time r_j. The integral flow of schedule S is $F(S) = \sum_{j\in J} F_j$. In contrast, the *fractional flow* can be seen as the flow on a per workload basis (instead of per job). More formally, if $p_j(t)$ denotes the work remaining on job j at time t, the fractional flow time of job j is $w_j \int_{r_j}^{\infty} \frac{p_j(t)}{p_j} dt$. Our goal is to find energy-efficient schedules that provide a good (low) flow. We consider two different ways to combine these conflicting goals. In the *budget setting*, we fix an *energy budget* $B \geq 0$ and seek the minimal (fractional or integral) flow achievable with this energy. In the *flow plus energy setting*, we want to minimize a linear combination $F(S) + \beta E(S)$ of energy and (fractional or integral) flow.

3 Hardness Results

This section proves NP-hardness for the problems `B-IDUA` and `B-IDWU`. The reductions are from the subset sum problem, where we are given n elements $a_1 \geq a_2 \geq \cdots \geq a_n$ with $a_i \in \mathbb{N}$ as well as a target value $A \in \mathbb{N}$ with $a_1 < A < \sum_{i=1}^{n} a_i$. The goal is to decide whether there is a subset $L \subseteq [n]$ such that $\sum_{i\in L} a_i = A$.

Basic Idea. For both reductions, we define for each element a_i a job set J_i such that jobs of different sets will not influence each other. Each J_i contains one low density job and one/several high density jobs. Starting from a *base schedule*, we choose the parameters such that investing roughly a_i energy into J_i improves its flow by roughly a_i. More precisely, when J_i gets a_i energy, additional energy can be used to decreases the flow at a rate $\gg 1/2$ per energy unit. Given substantially more or less energy, additional energy decreases the flow at a rate of only $1/2$. We achieve this by ensuring that at about a_i energy, the schedule switches from finishing the low density job after the high density jobs to finishing it before them. For an energy budget of A, we can define a target flow that is reached if and only if there is an $L \subseteq [n]$ such that $\sum_{i\in L} a_i = A$ (corresponding to job sets that are given about a_i extra energy).

Remarks. We assume a processor with two speeds $s_1 = 1$ and $s_2 = 2$ and power consumption rates $P_1 = 1$ and $P_2 = 4$. For an isolated job of weight w, this means that increasing a workload of x from speed s_1 to s_2 increases the energy by x and decreases the flow by $w \cdot \frac{x}{2}$. To ensure that jobs of different job sets do not influence each other, one can increase all release times of the job set J_i by the total workload of all previous job sets. For ease of exposition, we consider each job group J_i in isolation, and assume its first job is released at time 0. Due to space constraints, the reduction for `B-IDWU` is deferred to the full version.

3.1 Hardness of B-IDUA

For $i \in [i]$, we define a job set $J_i = \{(i,1),(i,2)\}$ of two unit weight jobs and set $\delta = \frac{1}{a_1 n^2}$. The release time of job $(i,1)$ is $r_{i1} = 0$ and its size is $p_{i1} = a_i$. The release time of job $(i,2)$ is $r_{i2} = a_i/2$ and its size is $p_{i2} = 2\delta a_i$.

Definition 1 (Base Schedule). *The* base schedule BS_i *schedules job* $(i,1)$ *at speed 1 and job* $(i,2)$ *at speed* 2*. It finishes job* $(i,1)$ *after* $(i,2)$*, has energy consumption* $E(\mathrm{BS}_i) = a_i + 4\delta a_i$*, and flow* $F(\mathrm{BS}_i) = a_i + 2\delta a_i$*.*

Note that BS_i is optimal for the energy budget $E(\mathrm{BS}_i)$. Consider an optimal schedule S for the jobs $J = \bigcup_{i=1}^{n} J_i$ (release times shifted such that they do not interfere) for the energy budget $B = \sum_{i=1}^{n} E(\mathrm{BS}_i) + A$. Let $L \subseteq [n]$ be such that $i \in L$ if and only if J_i gets at least $E(\mathrm{BS}_i) + a_i - 4\delta a_i = 2a_i$ energy in S.

Lemma 1. *S has flow at most $F = \sum_{i=1}^{n} F(\mathrm{BS}_i) - (\frac{1}{2}+\delta)A$ iff $\sum_{i \in L} a_i = A$.*

Proof. For the first direction, given that $\sum_{i \in L} a_i = A$, note that the schedule that gives each job set J_i with $i \in L$ exactly $E(\mathrm{BS}_i) + a_i$ energy and each J_i with $i \notin L$ exactly $E(\mathrm{BS}_i)$ energy adheres to the energy budget and has flow exactly F. For the other direction, consider $i \in [n]$, let E_i be the total energy used to schedule J_i in S, and let $\Delta_i = E_i - E(\mathrm{BS}_i)$ the additional energy used with respect to the base schedule. Then, for $i \notin L$, the flow of J_i is $F(\mathrm{BS}_i) - \frac{1}{2}\Delta_i$, yielding an average flow gain per energy unit of $1/2$. For $i \in L$, the flow gain per energy unit is 1 for the interval $[2a_i, 2a_i + 2\delta a_i)$ and $1/2$ otherwise. Thus, the maximum average flow gain is achieved for $E_i = 2a_i + 2\delta a_i$, where the energy usage is $E(\mathrm{BS}_i) + a_i - 2\delta a_i$ and the flow is $F(\mathrm{BS}_i) - a_i/2$. This yields a maximum average flow gain per energy unit of $\frac{a_i/2}{a_i - 2\delta a_i} = \frac{1}{2-4\delta}$. Using these observations, we now show that, if $\sum_{i \in L} a_i \neq A$, the schedule has either too much flow or uses too much energy. Let us distinguish two cases:

Case 1: $\sum_{i \in L} a_i < A$*:* Using $a_i, A \in \mathbb{N}$ and our observations, the flow decreases by at most (w.r.t. $\sum_{i=1}^{n} \mathrm{BS}_i$)

$$\frac{1}{2-4\delta} \sum_{i \in L} a_i + \frac{1}{2}\Big(A - \sum_{i \in L} a_i\Big) = \frac{1}{2}A + \frac{\delta}{1-2\delta} \sum_{i \in L} a_i \leq \frac{1}{2}A + \frac{\delta}{1-2\delta}(A-1) < \Big(\frac{1}{2}+\delta\Big)A.$$

The last inequality follows from $\delta = \frac{1}{a_1 n^2} < \frac{1}{2A}$.

Case 2: $\sum_{i \in L} a_i > A$*:* This implies $\sum_{i \in L} a_i \geq A + 1$. Note that even if all jobs $(i, 2)$ with $i \in \{1,2,...,n\}$ are run at speed 1 instead of speed 2, the total energy saved with respect to the base schedules is at most $\sum_{i=1}^{n} 2\delta a_i \leq \frac{2}{n}$. By this and the previous observations, the additional energy used by S with respect to the base schedules is at least $(1-4\delta)\sum_{i \in L} a_i - \frac{2}{n} \geq \sum_{i \in L} a_i - \frac{6}{n} \geq A + 1 - \frac{6}{n} > A$. □

Theorem 1. *`B-IDUA` is NP-hard.*

4 Polynomial Time Algorithms

In this section we provide polynomial time algorithms for `FE-IDUU`, `FE-ICUU`, and `FE-FCWA`. The algorithm for `FE-ICUU` generalizes and makes slight modifications to the algorithm in [2] to handle arbitrary power functions. We also provide a new, simple, combinatorial algorithm for `FE-IDUU`. While by the results of Section 5.1 we could use the algorithm for `FE-ICUU` to solve `FE-IDUU`, the algorithm we provide has the advantages of not having the numerical qualifications of the algorithm for `FE-ICUU`, as well as providing some additional insight into the open problem `FE-IDUA`. The algorithm for `FE-FCWA` generalizes and makes slight modifications to the algorithm in [4] to handle arbitrary power functions.

4.1 An Algorithm for FE-IDUU

Here we give a polynomial time algorithm for `FE-IDUU`. We describe the algorithm for two speeds; it is straightforward to generalize it to k speeds. The algorithm relies heavily upon the fact that, when jobs are of unit size, the optimal completion ordering is always FIFO (since any optimal schedule uses the SRPT (shortest remaining processing time) scheduling policy).[4] Before describing the algorithm, we provide the necessary optimality conditions in Lemma 2. They are based on the following definitions, capturing how jobs may affect each other.

[4] In fact, a slightly more general result yields an optimal `FE-IDUA` schedule given an optimal completion ordering.

Definition 2 (Lower Affection). *For a fixed schedule, a job j_1* lower affects *a job j_2 if there is an $\varepsilon > 0$ such that decreasing the speed of j_1 by any value in $(0,\varepsilon]$ increases the flow of j_2.*

Definition 3 (Upper Affection). *For a fixed schedule, a job j_1* upper affects *a job j_2 if there is some $\varepsilon > 0$ such that increasing the speed of j_1 by any value in $(0,\varepsilon]$ decreases the flow of j_2.*

Lemma 2. *Be S an optimal schedule and s_1 and s_2 consecutive speeds. Define $\alpha = \frac{P_2 - P_1}{s_2 - s_1}$ and $\kappa = -(P_1 - \alpha s_1) \geq 0$. For job j with (interpolated) speed $s_j \in [s_1, s_2]$: (a) $s_j > s_1 \Rightarrow j$ lower affects at least κ jobs, and (b) $s_j < s_2 \Rightarrow j$ upper affects at most $\kappa - 1$ jobs.*

Proof. We start with (a). To get a contradiction, assume $s_j > s_1$ but j lower affects less than κ jobs. Thus, for any $\varepsilon > 0$, increasing j's completion time by ε increases the flow of at most $\kappa - 1$ jobs by ε. If the resulting schedule is S'. For $t = \frac{1}{s_j}$, the energy from S to S' decreases by

$$\begin{aligned} &tP(1/t) - (t+\varepsilon)P(1/(t+\varepsilon))t(\alpha/t + P_1 - \alpha s_1) - (t+\varepsilon)(\alpha/(t+\varepsilon) + P_1 - \alpha s_1) \\ &= \alpha + tP_1 - t\alpha s_1 - \alpha - (t+\varepsilon)P_1 + (t+\varepsilon)\alpha s_1 = -\varepsilon(P_1 - \alpha s_1) = \kappa\varepsilon. \end{aligned}$$

So, the total change in the objective function is at most $(\kappa - 1)\varepsilon - \kappa\varepsilon < 0$, contradicting the optimality of S. Statement (b) follows similarly by decreasing the completion time of j by ε. □

Observation 2. *Consider two arbitrary jobs j and j' in an arbitrary schedule S.*
(a) If j upper affects $j' \neq j$ and j does not run at s_2, j' must run at s_1.
(b) While raising j's speed, the number of its lower and upper affections can only decrease.
(c) If j upper affects j', then changing the speed of j' will not change j's affection on j'.
(d) Assume j runs at speed s_j and upper affects m jobs. Then, in any schedule where j's speed is increased (and all other jobs remain unchanged), j lower affects at most m jobs.

Our algorithm GREEDYAFFECTION initializes each job with speed s_1. Consider jobs in order of release times and let j denote the current job. While j upper affects at least κ jobs and is not running at s_2, increase its speed. Otherwise, update j to the next job (or terminate if $j = n$).

Theorem 3. GREEDYAFFECTION *solves* `FE-IDUU` *in polynomial time.*

Proof. Assume A is not optimal and let O be an optimal schedule agreeing with A for the most consecutive job speeds (in order of release times). Let j be the first job that runs at a different speed and let s_A and s_O be the job's speeds in A and O. We consider two cases: If $s_A > s_O$, Observation 2(a) implies that every job that is upper affected by j in O other than j itself is run at s_1. Consider the time of A's execution when the speed of j was at s_O. Since A continued to raise j's speed, j upper affected at least κ jobs. Let J be this set of jobs. By Observation 2(c), j still upper affects all jobs $j' \in J$ in O. This contradicts the optimality of O (Lemma 2). For the second case, assume $s_A < s_O$. By Lemma 2, j upper affects less than κ jobs in A. When A stops raising j's speed, all jobs to the right run at s_1. Observations 2(b) and (d) imply that j lower affects less than κ jobs in O, contradicting O's optimality (Lemma 2). □

4.2 An Algorithm for FE-ICUU

In this subsection we show that `FE-ICUU` is in P. Essentially, it is possible to modify the algorithm from [2] to work with arbitrary power functions. The main alteration is that, for certain power functions that would yield differential equations too complicated for the algorithm to solve, we use binary search to find solutions to a these equations rather than solve the equations analytically.

Theorem 4. *There is a polynomial time algorithm for solving* `FE-ICUU`.

4.3 An Algorithm for FE-FCWA

This subsection shows that FE−FCWA is in P. The basic idea is to modify the algorithm from [4] to work with arbitrary power functions, under some mild assumptions. In order to maintain polynomial running time, the algorithm must efficiently find the next occurrence of certain *events*. In [4], this is done by analytically solving a series of differential equations, which are too complicated to solve for arbitrary power functions. Instead, our algorithm finds the occurrence of the next event by using binary search to "guess" its occurrence, and then (numerically) solve a (simpler) set of equations to determine if an event did, in fact, occur. Our only assumption is that it is possible to numerically find a solution to the involved equations.

Theorem 5. *There is a polynomial time algorithm for solving* FE−FCWA.

5 Equivalence Reductions

Here we provide the reductions to obtain the hardness and algorithmic results that are not proven explicitly. First, we reduce B−ICUU to FE−ICUU. Combined with the algorithm from Section 4.2, this shows that B−ICUU is in P. The second reduction is from any problem in the discrete power setting to the corresponding continuous variant. As a result, the hardness proofs from Section 3 for B−IDWU and B−IDUA imply that B−ICWU and B−ICUA are NP-hard. Our final reduction is from B−FCWA to FE−FCWA. As a result of the algorithm in Section 4.3, this shows that B−FCWA is in P.

5.1 Reducing B-ICUU to FE-ICUU

We show that, given an algorithm for the flow plus energy variant, we can solve the energy budget variant of ICUU. The basic idea is to modify the coefficient β in the flow plus energy objective until we find a schedule that fully utilizes the energy budget B. This schedule gives the minimum flow for B. The major technical hurdles to overcome are that the power function P may be non-differentiable, and may lead to multiple optimal flow plus energy schedules, each using different energies. Thus, we may not find a corresponding schedule for the given budget, even if there is one. To overcome this, we define the *affectance* ν_j of a job j. Intuitively, ν_j represents how many jobs' flows will be affected by a speed change of j. We show that a job's affectance is, in contrast to its energy and speed, unique across optimal schedules and changes continuously in β. This will imply that job speeds change continuously in β (i.e., for small enough changes, there are two optimal schedules with speeds arbitrarily close). We also give a continuous transformation process between any two optimal schedules. This eventually allows us to apply binary search to find the correct β.

Definitions & Notation. We start with some formal definitions for this section and a small overview of what they will be used for in the remainder. Definition 6 (affectance) will be most central to this section, as it will be shown in Lemma 3 and Corollary 1 to characterize optimal schedules. It uses the *subdifferential*[5] $\partial P(s)$ to handle non-differentiable power functions P.

Definition 4 (Total Weight of Lower/Upper Affection). *In any schedule, l_j and u_j are the total weight of jobs lower and upper affected by j, respectively (see Definitions 2 and 3).*

[5] Subdifferentials generalize the derivative of convex functions. $\partial P(s)$ is the set of slopes of lines through $(s, P(s))$ that lower bound P. It is closed, convex on the interior of P's domain, and non-decreasing if P is increasing [13].

Definition 5 (Job Group). *A* job group *is a maximal subset of jobs such that each job in the subset completes after the release of the next job. Let J_i denote the job group with the i-th earliest release time and W_i the total weight of J_i ($J_i = \emptyset$ and $W_i = 0$ if J_i does not exist). Job groups J_i and J_{i+1} are* consecutive *if the last job in J_i ends at the release time of the first job in J_{i+1}. We set the indicator $\zeta_i = 1$ if and only if J_{i+1} exists and J_i and J_{i+1} are consecutive.*

Definition 6 (Affectance Property). *The ith job group of a schedule satisfies the* affectance property *if either $\zeta_{i+1} = 0$ or the $i+1$st job group also satisfies the affectance property, and there exists $\mathcal{N}^i$ such that for all $v^i \in \mathcal{N}^i$ and $j \in J_i$*

$$v^i \in [0, \zeta_{i+1}(\nu^{i+1} + W_{i+1})], \tag{1}$$

$$v_j = v^i + u_j, \text{ and} \tag{2}$$

$$v_j = s_j d - P(s_j) \text{ for some } d \in \partial P(s). \tag{3}$$

Here, $\nu^i = \max \mathcal{N}^i$ if job group i exists, and $\nu^i = 0$ otherwise. A schedule satisfies the affectance property if all job groups in the schedule satisfy the affectance property.

Definition 7 (Affectance of a Job). *The set of speeds satisfying Eq.* (3) *for $v_j = \nu$ is $\mathcal{S}(\nu)$. For each job j in group i with the affectance property, the* affectance of job j *is $\nu_j = \nu^i + u_j$.*

Characterizing Optimal Schedules. We first prove that the affectance property characterizes optimal schedules. Lemma 3 shows that this property is necessary, Lemma 4 shows that affectance is unique across optimal schedules, and Corollary 1 shows that the affectance property is sufficient for optimality.

Lemma 3. *Any optimal schedule for* `FE-ICUU` *satisfies the affectance property.*

Lemma 4. *Let S_1 and S_2 be schedules with the affectance property and let ν_j^i denote the affectance of job j in the corresponding schedule. Then $\nu_j^1 = \nu_j^2$ for all j.*

Next, we show how to transform any schedule that has the affectance property into any other such schedule without changing the flow plus energy value. Together with Lemma 3, this immediately implies that the affectance property is sufficient for optimality (Corollary 1). Also, Lemma 3 is a nice algorithmic tool, as it allows us to find schedules "in between" any two optimal schedules with arbitrary precision. We will make use of that in the proof of Theorem 6.

Lemma 5. *Let S_1 and S_2 be schedules with the affectance property. We can transform S_1 to S_2 without changing its flow plus energy. All intermediate schedules satisfy the affectance property and we can make the speed changes between intermediate schedules arbitrarily small.*

Corollary 1. *Any schedule satisfying the affectance property is optimal.*

Binary Search Algorithm. We now provide the main technical result of this section, a polynomial time algorithm for `B-ICUU` based on any such algorithm for `FE-ICUU` (Theorem 6). In order to state the algorithm and its correctness, we need two more auxiliary lemmas. Lemma 6 proves that the affectance of jobs is continuous in β, while Lemma 7 does the same for job speeds.

Lemma 6. *For $\beta > 0$ and $\varepsilon > 0$, there exists $\delta > 0$ such that for all jobs j and $\beta' \in [\beta - \delta, \beta + \delta]$, any optimal* `FE-ICCU` *schedules S for β and S' for β' adhere to $\nu'_j \in [\nu_j - \varepsilon, \nu_j + \varepsilon]$.*

Lemma 7. *For $\beta > 0$ and $\varepsilon > 0$, there exists $\delta > 0$ such that for all jobs j and $\beta' \in [\beta-\delta,\beta+\delta]$, any optimal* FE−ICUU *schedules S for β and S' for β' adhere to $s'_j \in [s_j-\varepsilon, s_j+\varepsilon]$.*

Theorem 6. *Given a polynomial time algorithm for the continuous flow plus energy problem with unit size unit weight jobs, there is a polynomial time algorithm for the budget variant.*

Proof. Suppose we are given an energy budget B, and an algorithm to solve FE−ICUU. As we formally show in the proof of Theorem 8, the energy of optimal schedules increases as β decreases (even though we are considering here integral flow rather than fractional flow). Thus, the first step of the algorithm is to binary search over β until we find a schedule that fully utilizes B. If we find such a β, we are done (any optimal FE−ICUU schedule must minimize flow for the energy it consumes). Otherwise, we consider three cases:

Case 1: We find a β for which the optimal FE−ICUU schedule runs every job at the lowest speed used by any optimal schedule and uses $> B$ energy. Here, this lowest speed is (if it exists) the largest speed s such that for all $s' < s$ we have $\frac{P(s)}{s} \leq \frac{P(s')}{s'}$. In this case, no solution exists, since running a job at a lower speed increases its flow but does not decrease its energy.

Case 2: We find a β for which the optimal FE−ICUU schedule runs every job at the highest speed used by any optimal schedule and uses $\leq B$ energy. Here, this highest speed is (if it exists) the largest speed s such that for all $s' > s$ we have $P(s') = \infty$. In this case, β yields the optimal budget solution, since running any job at a higher speed uses infinite energy.

Case 3: There is $\varepsilon > 0$ such that for any β, the computed optimal FE−ICUU schedule uses at least $B+\varepsilon$ or at most $B-\varepsilon$ energy. Since job speeds are continuous in β (Lemma 7) and the energy increases as β decreases, we know that there is some β such that the corresponding FE−ICUU solutions contain schedules using both $B+\varepsilon_1$ energy and $B-\varepsilon_2$ energy ($\varepsilon_1, \varepsilon_2 > 0$). Fix such a β and let S_1 and S_2 be the corresponding schedules using $B-\varepsilon_1$ and $B+\varepsilon_2$, respectively. By Lemma 5, we can continuously change the speeds (and, thus, energy) of S_1 to obtain S_2. During this process, we obtain an intermediate optimal FE−ICUU schedule that uses exactly B energy. As described above, this schedule is also optimal for B−ICUU. □

5.2 Reducing the Discrete to the Continuous Setting

The main result of this subsection is a reduction from the discrete to the continuous setting. Using mild computational power assumptions, Theorem 7 shows how to use an algorithm for the continuous variant of one of our problems (⋆−⋆C⋆⋆) to solve the corresponding discrete variant (⋆−⋆D⋆⋆). It is worth noting that our reduction makes use of arbitrary continuous power functions (especially power functions with a maximum speed).

Theorem 7. *Given a polynomial time algorithm for any budget or flow plus energy variant in the continuous setting, there is a polynomial time algorithm for the discrete variant.*

5.3 Reducing from Budget to Flow Plus Energy for Fractional Flow

This subsection gives a reduction from the budget to the flow plus energy objective. The reduction given in Theorem 8 is for fractional flow, assumes the most general setting (weighted jobs of arbitrary size), and preserves unit size and unit weight jobs, making it applicable to reduce B−FC⋆⋆ to FE−FC⋆⋆.

Theorem 8. *Given a polynomial time algorithm for the budget variant and fractional flow, there is a polynomial time algorithm for the corresponding flow plus energy variant.*

Bibliography

[1] S. Albers. Energy-efficient algorithms. *Commun. of the ACM*, 53(5):86–96, 2010.

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

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

[4] A. Antoniadis, N. Barcelo, M. Consuegra, P. Kling, M. Nugent, K. Pruhs, and M. Scquizzato. Efficient computation of optimal energy and fractional weighted flow trade-off schedules. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 63–74, 2014.

[5] N. Bansal, H.-L. Chan, T. W. Lam, and L.-K. Lee. Scheduling for speed bounded processors. In *International Conference on Automata, Languages, and Programming (ICALP)*, pages 409–420, 2008.

[6] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

[7] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2):18:1–18:14, 2013.

[8] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2(6): 245–252, 1999.

[9] P. Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Appl. Math.*, 103(1-3):21–32, July 2000.

[10] N. Barcelo, D. Cole, D. Letsios, M. Nugent, and K. Pruhs. Optimal energy trade-off schedules. *Sustainable Computing: Informatics and Systems*, 3:207–217, 2013.

[11] S.-H. Chan, T. W. Lam, and L.-K. Lee. Non-clairvoyant speed scaling for weighted flow time. In *European Symposium on Algorithms (ESA), Part I*, pages 23–35, 2010.

[12] N. R. Devanur and Z. Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1123–1140, 2014.

[13] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2001.

[14] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Preemptive scheduling of uniform machines subject to release dates. In P. H. R., editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[15] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *European Symposium on Algorithms (ESA)*, pages 647–659, 2008.

[16] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *International Conference on Automata, Languages, and Programming (ICALP)*, pages 745–756, 2013.

[17] K. Pruhs, P. Uthaisombut, and G. J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.