Energy-Efficient Algorithms for Flow Time Minimization

Susanne Albers[1] and Hiroshi Fujiwara[2,*]

[1] Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee 79, 79110 Freiburg, Germany
salbers@informatik.uni-freiburg.de

[2] Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Japan
fujiwara@lab2.kuis.kyoto-u.ac.jp

Abstract. We study scheduling problems in battery-operated computing devices, aiming at schedules with low total energy consumption. While most of the previous work has focused on finding feasible schedules in deadline-based settings, in this paper we are interested in schedules that guarantee good response times. More specifically, our goal is to schedule a sequence of jobs on a variable speed processor so as to minimize the total cost consisting of the power consumption and the total flow time of all the jobs. We first show that when the amount of work, for any job, may take an arbitrary value, then no online algorithm can achieve a constant competitive ratio. Therefore, most of the paper is concerned with unit-size jobs. We devise a deterministic constant competitive online algorithm and show that the offline problem can be solved in polynomial time.

1 Introduction

Embedded systems and portable devices play an ever-increasing role in every day life. Prominent examples are mobile phones, palmtops and laptop computers that are used by a significant fraction of the population today. Many of these devices are battery-operated so that effective power management strategies are essential to guarantee a good performance and availability of the systems. The microprocessors built into these devices can typically perform tasks at different speeds – the higher the speed, the higher the power consumption is. As a result, there has recently been considerable research interest in dynamic speed scaling strategies; we refer the reader to [1, 2, 3, 7, 10, 13] for a selection of the papers that have been published in algorithms conferences.

Most of the previous work considers a scenario where a sequence of jobs, each specified by a release time, a deadline and an amount of work that must be performed to complete the task, has to be scheduled on a single processor. The processor may run at variable speed. At speed s, the power consumption is

[*] This work was done while visiting the University of Freiburg.

$P(s) = s^\alpha$ per time unit, where $\alpha > 1$ is a constant. The goal is to find a feasible schedule such that the total power consumption over the entire time horizon is as small as possible. While this basic framework gives insight into effective power conservation, it ignores the important aspect that users typically expect good response times for their jobs. Furthermore, in many computational systems, jobs are not labeled with deadlines. For example, operating systems such as Window and Unix installed on laptops do not employ deadline-based scheduling.

Therefore, in this paper, we study algorithms that minimize energy usage and at the same time guarantee good response times. In the scientific literature, response time is modeled as *flow time*. The flow time of a job is the length of the time interval between the release time and the completion time of the job. Unfortunately, energy minimization and flow time minimization are orthogonal objectives. To save energy, the processor should run at low speed, which yields high flow times. On the other hand, to ensure small flow times, the processor should run at high speed, which results in a high energy consumption. In order to overcome this conflict, Pruhs et al. [10] recently studied the problem of minimizing the average flow time of a sequence of jobs when a *fixed amount of energy* is available. They presented a polynomial time offline algorithm for unit-size jobs. However, it is not clear how to handle the online scenario where jobs arrival times are unknown.

Instead, in this paper, we propose a different approach to integrate energy and flow time minimization: We seek schedules that minimize the total cost consisting of the power consumption and the flow times of jobs. More specifically, a sequence of jobs, each specified by an amount of work, arrives over time and must be scheduled on one processor. Preemption of jobs is not allowed. The goal is to dynamically set the speed of the processor so as to minimize the sum of (a) the total power consumption and (b) the total flow times of all the jobs. Such combined objective functions have been studied for many other bicriteria optimization problems with orthogonal objectives. The papers [5, 8], e.g., consider a TCP acknowledgement problem, minimizing the sum of acknowledgement costs and acknowledgement delays incurred for data packets. In [6] the authors study network design and minimize the total hardware and QoS costs. More generally, in the classical facility location problem, one minimizes the sum of the facility installation and total client service costs, see [4, 9] for surveys.

For our energy/flow-time minimization problem, we are interested in both online and offline algorithms. Following [11], an online algorithm A is said to be c-competitive if there exists a constant a such that, for all job sequences σ, the total cost $A(\sigma)$ satisfies $A(\sigma) \le c \cdot \mathrm{OPT}(\sigma) + a$, where $\mathrm{OPT}(\sigma)$ is the cost of an optimal offline algorithm.

Previous work: In their seminal paper, Yao et al. [13] introduced the basic problem of scheduling a sequence of jobs, each having a release time, a deadline and a certain workload, so as to minimize the energy usage. Here, preemption of jobs is allowed. Yao et al. showed that the offline problem can be solved optimally in polynomial time and presented two online algorithms called *Average Rate* and *Optimal Available*. They analyzed *Average Rate*, for $\alpha \ge 2$, and proved an upper bound of $2^\alpha\alpha^\alpha$ and a lower bound of α^α on the competitiveness. Bansal et al. [2] studied *Optimal Available* and showed that its competitive ratio is exactly α^α. Furthermore, they developed a new algorithm that achieves a competitiveness of $2(\alpha/(\alpha-1))^\alpha e^\alpha$ and proved that any randomized online algorithm has a performance ratio of at least $\Omega((4/3)^\alpha)$.

Irani et al. [7] studied an extended scenario where the processor can be put into a low-power sleep state when idle. They gave an offline algorithm that achieves a 3-approximation and developed a general strategy that transforms an online algorithm for the setting without sleep state into an online algorithm for the setting with sleep state. They obtain constant competitive online algorithms, but the constants are large. For the famous cube root rule $P(s) = s^3$, the competitive ratio is 540. The factor can be reduced to 84, see [2]. Settings with several sleep states were considered in [1]. Speed scaling to minimize the maximum temperature of a processor was addressed in [2,3].

As mentioned above, Pruhs et al. [10] study the problem of minimizing the average flow time of jobs given a fixed amount of energy. For unit-size jobs, they devise a polynomial time algorithm that simultaneously computes, for each possible energy level, the schedule with smallest average flow time.

Our contribution: We investigate the problem of scheduling a sequence of n jobs on a variable speed processor so as to minimize the total cost consisting of the power consumption and the flow times of jobs. We first show that when the amount of work, for any job, may take an arbitrary value, then any deterministic online algorithm has a competitive ratio of at least $\Omega(n^{1-1/\alpha})$. This result implies that speed scaling does not help to overcome bad scheduling decisions: It is well-known that in standard scheduling, no online algorithm for flow time minimization can be better than $\Omega(n)$-competitive. Our lower bound, allowing speed scaling, is almost as high.

Because of the $\Omega(n^{1-1/\alpha})$ lower bound, most of our paper is concerned with unit-size jobs. We develop a deterministic phase-based online algorithm that achieves a constant competitive ratio. The algorithm is simple and requires scheduling decisions to be made only every once in a while, which is advantageous in low-power devices. Initially, the algorithm computes a schedule for the first batch of jobs released at time 0. While these jobs are being processed, the algorithm collects the new jobs that arrive in the meantime. Once the first batch of jobs is finished, the algorithm computes a schedule for the second batch. This process repeats until no more jobs arrive. Within each batch the processing speeds are easy to determine. When there are i unfinished jobs in the batch, the speed is set to $\sqrt[\alpha]{i/c}$, where c is a constant that depends on the value of α. We prove that the competitive ratio of our algorithm is upper bounded by $8.3e(1+\Phi)^\alpha$, where $\Phi = (1+\sqrt{5})/2 \approx 1.618$ is the Golden Ratio. We remark that a phase-based scheduling algorithm was also used in makespan minimization on parallel machines [12]. However, for our problem, the scheduling strategy within the phases and the analysis techniques employed are completely different.

Furthermore, in this paper we develop a polynomial time algorithm for computing an optimal offline schedule. We would like to point out that we could use the algorithm by Pruhs et al. [10], but this would yield a rather complicated algorithm for our problem. Instead, we design a simple, direct algorithm based on dynamic programming. Our approach can also be used to address the problem of Pruhs et al., i.e. we are able to determine a schedule with minimum flow time given a fixed amount of enery. This can be seen as an additional advantage of our new objective function.

2 Preliminaries

Consider a sequence of jobs $\sigma = \sigma_1, \ldots, \sigma_n$ which are to be scheduled on one processor. Job σ_i is released at time r_i and requires p_i CPU cycles. We assume $r_1 = 0$ and $r_i \leq r_{i+1}$, for $i = 1, \ldots, n-1$. A schedule $\mathcal{S}$ specifies, for each job σ_i, a time interval I_i and a speed s_i such that σ_i is processed at speed s_i continuously, without interruption, throughout I_i. Let $P(s) = s^\alpha$ be the power consumption per time unit of the CPU depending on s. The constant $\alpha > 1$ is a real number. As $P(s)$ is convex, we may assume w.l.o.g. that each σ_i is processed at a constant speed s_i. A schedule $\mathcal{S}$ is feasible if, for any i, interval I_i starts no earlier than r_i, and the processing requirements are met, i.e. $p_i = s_i|I_i|$. Here $|I_i|$ denotes the length of I_i. Furthermore, in a feasible schedule $\mathcal{S}$ the intervals I_i must be non-overlapping. The energy consumption of $\mathcal{S}$ is $E(\mathcal{S}) = \sum_{i=1}^n P(s_i)|I_i|$. For any i, let c_i be the completion time of job i, i.e. c_i is equal to the end of I_i. The flow time of job i is $f_i = c_i - r_i$ and the flow time of $\mathcal{S}$ is given by $F(\mathcal{S}) = \sum_{i=1}^n f_i$. We seek schedules $\mathcal{S}$ that minimize the sum $g(\mathcal{S}) = E(\mathcal{S}) + F(\mathcal{S})$.

3 Arbitrary Size Jobs

We show that if the jobs' processing requirements may take arbitrary values, then no online algorithm can achieve a bounded competitive ratio. The proof of the following theorem is omitted due to space constraints.

Theorem 1. *The competitive ratio of any deterministic online algorithm is* $\Omega(n^{1-1/\alpha})$ *if the processing requirements* $p_1, \ldots, p_n$ *may take arbitrary values.*

4 An Online Algorithm for Unit-Size Jobs

In this section we study the case that the processing requirements of all jobs are the same, i.e. $p_i = 1$, for all jobs. We develop a deterministic online algorithm that achieves a constant competitive ratio, for all α. The algorithm is called *Phasebal* and aims at balancing the incurred power consumption with the generated flow time. If α is small, then the ratio is roughly $1 : \alpha - 1$. If α is large, then the ratio is $1 : 1$. As the name suggests, the algorithm operates in phases.

Let n_1 be the number of jobs that are released initially at time $t = 0$. In the first phase *Phasebal* processes these jobs in an optimal or nearly optimal way, ignoring jobs that may arrive in the meantime. More precisely, the speed sequence for the n_1 jobs is $\sqrt[\alpha]{n_1/c}$, $\sqrt[\alpha]{(n_1-1)/c}, \ldots$, $\sqrt[\alpha]{1/c}$, i.e. the j-th of these n_1 jobs is executed at speed $\sqrt[\alpha]{(n_1-j+1)/c}$ for $j = 1, \ldots, n_1$. Here c is a constant that depends on α. Let n_2 be the number of jobs that arrive in phase 1. *Phasebal* processes these jobs in a second phase. In general, in phase i *Phasebal* schedules the n_i jobs that arrived in phase $i-1$ using the speed sequence $\sqrt[\alpha]{(n_i-j+1)/c}$, for $j = 1, \ldots, n_i$. Again, jobs that arrive during the phase are ignored until the end of the phase. A formal description of the algorithm is as follows.

Algorithm Phasebal: If $\alpha < (19+\sqrt{161})/10$, then set $c := \alpha - 1$; otherwise set $c := 1$. Let n_1 be the number of jobs arriving at time $t = 0$ and set $i = 1$. While $n_i > 0$, execute the following two steps: (1) For $j = 1, \ldots, n_i$, process the j-th job using a speed of $\sqrt[\alpha]{(n_i-j+1)/c}$. We refer to this entire time interval as phase i. (2) Let n_{i+1} be the number of jobs that arrive in phase i and set $i := i + 1$.

Theorem 2. *Phasebal has a competitiveness of at most* $(1+\Phi)(1+\Phi^{\frac{\alpha}{(2\alpha-1)}})^{(\alpha-1)} \frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}} \min\{\frac{5\alpha-2}{2\alpha-1}, \frac{4}{2\alpha-1} + \frac{4}{\alpha-1}\}$, *where* $\Phi = (1+\sqrt{5})/2 \approx 1.618$.

Before proving Theorem 2, we briefly discuss the competitiveness. We first observe that $\frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}} \leq e\alpha$. Moreover, $\frac{\alpha(5\alpha-2)}{2\alpha-1}$ is increasing in α, while $\frac{4\alpha}{2\alpha-1} + \frac{4\alpha}{\alpha-1}$ is decreasing in α. Standard algebraic manipulations show that the latter two expressions are equal for $\alpha_0 = (19+\sqrt{161})/10$. Thus, the competitive ratio is upper bounded by $(1+\Phi)^\alpha e \frac{\alpha_0(5\alpha_0-2)}{2\alpha_0-1} < (1+\Phi)^\alpha e \cdot 8.22$.

In the remainder of this section we will analyze *Phasebal*. The global analysis consists of two cases. We will first address $c = 1$ and then $c = \alpha - 1$. In each case we first upper bound the total cost incurred by *Phasebal* and then lower bound the cost of an optimal schedule. In the case $c = 1$ we will consider a pseudo-optimal algorithm that operates with similar speeds as *Phasebal*. We will prove that the cost of such a pseudo-optimal algorithm is at most a factor of 2 away from the true optimum. In any case we will show that an optimal or pseudo-optimal algorithm finishes jobs no later than *Phasebal*. This property will be crucial to determine the time intervals in which optimal schedules process jobs and to lower bound the corresponding speeds. These speed bounds will then allow us to estimate the optimal cost and to finally compare it to the online cost.

Let $t_0 = 0$ and t_i be the time when phase i ends, i.e. the n_i jobs released during phase $i-1$ (released initially, if $i = 1$) are processed in the time interval $[t_{i-1}, t_i)$, which constitutes phase i. Given a job sequence σ, let $\mathcal{S}_{PB}$ be the schedule of *Phasebal* and let $\mathcal{S}_{OPT}$ be an optimal schedule.

Case 1: $\mathbf{c = 1}$ We start by analyzing the cost and time horizon of $\mathcal{S}_{PB}$. Suppose that there are k phases, i.e. no new jobs arrive in phase k. In phase i the algorithm needs $1/\sqrt[\alpha]{n_i-j+1}$ time units to complete the j-th job. Thus the power consumption in the phase is

$$\sum_{j=1}^{n_i}(\sqrt[\alpha]{n_i-j+1})^\alpha/\sqrt[\alpha]{n_i-j+1} = \sum_{j=1}^{n_i}(n_i-j+1)^{1-1/\alpha}$$
$$\leq \tfrac{\alpha}{2\alpha-1}(n_i^{2-1/\alpha}-1)+n_i^{1-1/\alpha}.$$

The length of phase i is

$$T(n_i) = \sum_{j=1}^{n_i} 1/\sqrt[\alpha]{n_i-j+1} \leq \tfrac{\alpha}{\alpha-1}n_i^{1-1/\alpha}. \qquad (1)$$

As for the flow time, the n_i jobs scheduled in the phase incur a flow time of

$$\sum_{j=1}^{n_i}(n_i-j+1)/\sqrt[\alpha]{n_i-j+1} \leq \tfrac{\alpha}{2\alpha-1}(n_i^{2-1/\alpha}-1)+n_i^{1-1/\alpha},$$

while the n_{i+1} jobs released during the phase incur a flow time of at most n_{i+1} times the length of the phase. We obtain

$$g(\mathcal{S}_{PB}) \leq \sum_{i=1}^{k}(\tfrac{2\alpha}{2\alpha-1}(n_i^{2-1/\alpha}-1)+2n_i^{1-1/\alpha}) + \sum_{i=1}^{k-1} n_{i+1}\tfrac{\alpha}{\alpha-1}n_i^{1-1/\alpha}.$$

The second sum is bounded by $\sum_{i=1}^{k-1}\frac{\alpha}{\alpha-1}\max\{n_i,n_{i+1}\}^{2-1/\alpha} \leq \sum_{i=1}^{k}\frac{2\alpha}{\alpha-1}n_i^{2-1/\alpha}$ and we conclude

$$g(\mathcal{S}_{PB}) \leq 2\sum_{i=1}^{k}(\tfrac{\alpha}{2\alpha-1}(n_i^{2-1/\alpha}-1)+n_i^{1-1/\alpha}+\tfrac{\alpha}{\alpha-1}n_i^{2-1/\alpha}). \qquad (2)$$

We next lower bound the cost of an optimal schedule. As mentioned before, it will be convenient to consider a pseudo-optimal schedule $\mathcal{S}_{POPT}$. This is the best schedule that satisfies the constraint that, at any time, if there are ℓ *active* jobs, then the processor speed is at least $\sqrt[\alpha]{\ell}$. We call a job active if it has arrived but is not yet finished. In the next lemma we show that the objective function value $g(\mathcal{S}_{POPT})$ is not far from the true optimum $g(\mathcal{S}_{OPT})$.

Lemma 1. *For any job sequence, $g(\mathcal{S}_{POPT}) \leq 2g(\mathcal{S}_{OPT})$.*

Proof. Consider the optimal schedule $g(\mathcal{S}_{OPT})$. We may assume w.l.o.g. that in this schedule the speed only changes when a jobs gets finished of new jobs arrive. We partition the time horizon of $\mathcal{S}_{OPT}$ into a sequence of intervals $I_1,\ldots,I_m$ such that, for any such interval, the number of active jobs does not change. Let $E(I_i)$ and $F(I_i)$ be the energy consumption and flow time, respectively, generated in I_i, $i=1,\ldots,m$. We have $E(I_i)=s_i^\alpha\delta_i$ and $F(I_i)=\ell_i\delta_i$, where s_i is the speed, ℓ_i is the number of active jobs in I_i and δ_i is the length of I_i. Clearly $g(\mathcal{S}_{OPT})=\sum_{i=1}^{m}(E(I_i)+F(I_i))$.

Now we change $\mathcal{S}_{OPT}$ as follows. In any interval I_i with $s_i<\sqrt[\alpha]{\ell_i}$ we increase the speed to $\sqrt[\alpha]{\ell_i}$, incurring an energy consumption of $\ell_i\delta_i$, which is equal to

$F(I_i)$ in original schedule $\mathcal{S}_{OPT}$. In this modification step, the flow time of jobs can only decrease. Because of the increased speed, the processor may run out of jobs in some intervals. Then the processor is simply idle. We obtain a schedule whose cost is bounded by $\sum_{i=1}^{m}(E(I_i) + 2F(I_i)) \leq 2g(\mathcal{S}_{OPT})$ and that satisfies the constraint that the processor speed it at least $\sqrt[\alpha]{\ell}$ in intervals with ℓ active job. Hence $g(\mathcal{S}_{POPT}) \leq 2g(\mathcal{S}_{OPT})$. □

The next lemma shows that in $\mathcal{S}_{POPT}$ jobs finish no later than in $\mathcal{S}_{PB}$.

Lemma 2. *For $c = 1$, in $\mathcal{S}_{POPT}$ the n_1 jobs released at time t_0 are finished by time t_1 and the n_i jobs released during phase $i-1$ are finished by time t_i, for $i = 2, \dots, k$.*

Proof. We show the lemma inductively. As for the n_1 jobs released at time t_0, the schedule $\mathcal{S}_{POPT}$ processes the j-th of these jobs at a speed of at least $\sqrt[\alpha]{n_1 - j + 1}$ because there are at least $n - j + 1$ active jobs. Thus the n_1 jobs are completed no later than $\sum_{j=1}^{n_1} 1/\sqrt[\alpha]{n_1 - j + 1}$, which is equal to the length of the first phase, see (1).

Now suppose that jobs released by time t_{i-1} are finished by time t_i and consider the n_{i+1} jobs released in phase i. At time t_i there are at most these n_{i+1} jobs unfinished. Let $\overline{n}_{i+1}$ be the actual number of active jobs at that time. Again, the j-th of these jobs is processed at a speed of at least $(\overline{n}_{i+1} - j + 1)^{1/\alpha}$ so that the execution of these $\overline{n}_{i+1}$ jobs ends no later than $\sum_{j=1}^{\overline{n}_{i+1}}(\overline{n}_{i+1} - j + 1)^{-1/\alpha}$ and this sum is not larger than the length of phase $i+1$, see (1). □

Lemma 3. *If a schedule has to process ℓ jobs during a time period of length $T \leq \ell\sqrt[\alpha]{\alpha - 1}$, then its total cost is at least $FLAT(\ell, T) \geq (\ell/T)^{\alpha}T + T$.*

The proof is omitted.

Lemma 4. *For $\alpha \geq 2$, there holds $g(\mathcal{S}_{POPT}) \geq C^{1-\alpha}(1+\Phi)^{-1}(1+\Phi^{\alpha/(2\alpha-1)})^{1-\alpha}$ $\sum_{i=1}^{k} n_i^{2-1/\alpha} + \sum_{i=1}^{k} T(n_i)$, where $C = \alpha/(\alpha-1)$ and $\Phi = (1+\sqrt{5})/2$.*

Proof. By Lemma 2, for $i \geq 2$, the n_i jobs arriving in phase $i-1$ are finished by time t_i in $\mathcal{S}_{POPT}$. Thus $\mathcal{S}_{POPT}$ processes these jobs in a window of length at most $T(n_{i-1}) + T(n_i)$. Let $T'(n_i) = \min\{T(n_{i-1}) + T(n_i), n_i\sqrt[\alpha]{\alpha - 1}\}$. Applying Lemma 3, we obtain that the n_i jobs incur a cost of at least

$$\begin{aligned}\frac{n_i^{\alpha}}{(T'(n_i))^{\alpha-1}} + T'(n_i) &\geq \frac{n_i^{\alpha}}{(T(n_{i-1}) + T(n_i))^{\alpha-1}} + T'(n_i)\\ &\geq \frac{n_i^{\alpha}}{(T(n_{i-1}) + T(n_i))^{\alpha-1}} + T(n_i).\end{aligned}$$

The last inequality holds because $T(n_i) \leq n_i \leq n_i\sqrt[\alpha]{\alpha - 1}$, for $\alpha \geq 2$ and hence $T'(n_i) \geq T(n_i)$. Similarly, for the n_1 jobs released at time $t = 0$, the cost it at least $n_1^{\alpha}/(T(n_1))^{\alpha-1} + T(n_1)$. Summing up, the total cost of $\mathcal{S}_{POPT}$ is at least

$$\frac{n_1^{\alpha}}{(T(n_1))^{\alpha-1}} + \sum_{i=2}^{k} \frac{n_i^{\alpha}}{(T(n_{i-1}) + T(n_i))^{\alpha-1}} + \sum_{i=1}^{k} T(n_i).$$

In the following we show that the first two terms in the above expression are at least $C^{1-\alpha}(1+\Phi)^{-1}(1+\Phi^{\alpha/(2\alpha-1)})^{1-\alpha}\sum_{i=1}^{k} n^{2-1/\alpha}$, which establishes the lemma to be proven. Since $T(n_i) \le Cn_i^{1-1/\alpha}$, it suffices to show

$$(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{\alpha-1}\left(\frac{n_1^\alpha}{\left(n_1^{1-1/\alpha}\right)^{\alpha-1}}+\sum_{i=2}^{k}\frac{n_i^\alpha}{\left(n_{i-1}^{1-1/\alpha}+n_i^{1-1/\alpha}\right)^{\alpha-1}}\right) \ge \sum_{i=1}^{k} n_i^{2-1/\alpha}. \quad (3)$$

To this end we partition the sequence of job numbers $n_1,\ldots,n_k$ into subsequences such that, within each subsequence, $n_i \ge \Phi^{\alpha/(2\alpha-1)}n_{i+1}$. More formally, the first subsequence starts with index $b_1 = 1$ and ends with the smallest index e_1 satisfying $n_{e_1} < \Phi^{\alpha/(2\alpha-1)}n_{e_1+1}$. Suppose that $l-1$ subsequences have been constructed. Then the l-st sequence starts at index $b_l = e_{l-1}+1$ and ends with the smallest index $e_l \ge b_l$ such that $n_{e_l} < \Phi^{\alpha/(2\alpha-1)}n_{e_l+1}$. The last subsequence ends with index k.

We will prove (3) by considering the individual subsequences. Since within a subsequence $n_{i+1} \le n_i\Phi^{-\alpha/(2\alpha-1)}$, we have $n_{i+1}^{2-1/\alpha} \le n_i^{2-1/\alpha}/\Phi$. Therefore, for any subsequence l, using the limit of the geometric series

$$\sum_{i=b_l}^{e_l} n_i^{2-1/\alpha} \le n_{b_l}^{2-1/\alpha}/(1-1/\Phi) = (1+\Phi)n_{b_l}^{2-1/\alpha}, \quad (4)$$

which upper bounds terms on the right hand side of (3). As for the left hand side of (3), we have for the first subsequence,

$$(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{\alpha-1}\left(\frac{n_1^\alpha}{\left(n_1^{1-1/\alpha}\right)^{\alpha-1}}+\sum_{i=2}^{e_1}\frac{n_i^\alpha}{\left(n_{i-1}^{1-1/\alpha}+n_i^{1-1/\alpha}\right)^{\alpha-1}}\right) \ge (1+\Phi)n_1^{2-1/\alpha}.$$

For any other subsequence l, we have

$$\begin{aligned}
&(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{\alpha-1}\sum_{i=b_l}^{e_l}\frac{n_i^\alpha}{\left(n_{i-1}^{1-1/\alpha}+n_i^{1-1/\alpha}\right)^{\alpha-1}}\\
&\ge (1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{\alpha-1}\frac{n_{b_l}^\alpha}{\left(n_{b_l-1}^{1-1/\alpha}+n_{b_l}^{1-1/\alpha}\right)^{\alpha-1}}\\
&\ge (1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{\alpha-1}\frac{n_{b_l}^\alpha}{\left((\Phi^{(\alpha-1)/(2\alpha-1)}+1)n_{b_l}^{1-1/\alpha}\right)^{\alpha-1}}\\
&\ge (1+\Phi)n_{b_l}^{2-1/\alpha}.
\end{aligned}$$

The second to last inequality holds because n_{b_l-1} and n_{b_l} belong to different subsequences and hence $n_{b_l-1} < \Phi^{\alpha/(2\alpha-1)} n_{b_l}$. The above inequalities together with (4) imply (3). □

Lemma 5. *For $\alpha \geq 2$ and $c = 1$, the competitive ratio of Phasebal is at most* $(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{(\alpha-1)} \frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}} (\frac{4}{2\alpha-1} + \frac{4}{\alpha-1})$.

Proof. Using (2) as well as Lemmas 1 and 4 we obtain that the competitive ratio of *Phasebal* is bounded by

$$(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{(\alpha-1)} \frac{4\sum_{i=1}^{k}((\frac{\alpha}{2\alpha-1}+\frac{\alpha}{\alpha-1})n_i^{2-1/\alpha} + n^{1-1/\alpha})}{\sum_{i=1}^{k}((\frac{\alpha}{\alpha-1})^{1-\alpha} n_i^{2-1/\alpha} + T(n_i))}.$$

Considering the terms of order $n^{2-1/\alpha}$, we obtain the performance ratio we are aiming at. It remains to show that $n_i^{1-1/\alpha}/T(n_i)$ does not violate this ratio. Note that $T(n_i) \geq 1$. Thus, if $n_i^{1-1/\alpha} \leq 2$ we have

$$n_i^{1-1/\alpha}/T(n_i) \leq 2 \leq 4(\tfrac{\alpha}{\alpha-1})^{\alpha-1}(\tfrac{\alpha}{2\alpha-1} + \tfrac{\alpha}{\alpha-1}). \tag{5}$$

If $n_i^{1-1/\alpha} > 2$, then we use the fact that $T(n_i) = \sum_{j=1}^{n_i} 1/\sqrt[\alpha]{n_i - j + 1} \geq \frac{\alpha}{\alpha-1}((n_i+1)^{1-1/\alpha} - 1) \geq \frac{1}{2}\frac{\alpha}{\alpha-1} n_i^{1-1/\alpha}$ and we can argue as in (5), since $(\alpha - 1)/\alpha < 1$. □

Case 2: $\mathbf{c} = \alpha - \mathbf{1}$ The global structure of the analysis is the same as in the case $c = 1$ but some of the calculations become more involved. Moreover, with respect to the optimum cost, we will consider the true optimum rather than the cost of a pseudo-optimal algorithm.

We start again by analyzing the cost and time of *Phasebal*. As before we assume that there are k phases. In phase i, *Phasebal* uses $1/\sqrt[\alpha]{(n_i - j + 1)/(\alpha - 1)}$ time units to process the j-th job. This yields a power consumption of

$$\sum_{j=1}^{n_i} \left(\frac{n_i - j + 1}{\alpha - 1}\right)^{1-1/\alpha} \leq C_E(n_i^{2-1/\alpha} - 1) + (\alpha - 1)^{1/\alpha - 1} n_i^{1-1/\alpha}$$

with $C_E = (\alpha - 1)^{\frac{1}{\alpha}-1}\frac{\alpha}{2\alpha-1}$. The phase length is $T(n_i) = \sum_{j=1}^{n_i} 1/\left(\frac{n_i-j+1}{\alpha-1}\right)^{1/\alpha}$. Here we have

$$C_T((n_i+1)^{1-1/\alpha} - 1) < T(n_i) < C_T(n_i^{1-1/\alpha} - 1/\alpha) \tag{6}$$

with $C_T = \alpha(\alpha - 1)^{\frac{1}{\alpha}-1}$. In phase i the n_i jobs processed during the phase incur a flow time of

$$\begin{aligned}\sum_{j=1}^{n_i}(n_i - j + 1)/\left(\frac{n_i - j + 1}{\alpha - 1}\right)^{1/\alpha} &= (\alpha - 1)^{1/\alpha}\sum_{j=1}^{n_i}(n_i - j + 1)^{1-1/\alpha} \\ &\leq C_F(n_i^{2-1/\alpha} - 1) + (\alpha - 1)^{1/\alpha} n_i^{1-1/\alpha}\end{aligned}$$

with $C_F = (\alpha-1)^{\frac{1}{\alpha}}\frac{\alpha}{2\alpha-1}$, while the n_{i+1} jobs arriving in the phase incur a cost of at most $n_{i+1}T(n_i)$. We obtain

$$g(\mathcal{S}_{PB}) \leq (C_E+C_F)\sum_{i=1}^{k}(n_i^{2-1/\alpha}-1)+2C_T\sum_{i=1}^{k}n_i^{2-1/\alpha}+\alpha(\alpha-1)^{1/\alpha-1}\sum_{i=1}^{k}n_i^{1-1/\alpha}. \tag{7}$$

We next lower bound the cost of an optimal schedule. Again we call a job *active* if it has arrived but is still unfinished. The proofs of the next three lemmas are omitted.

Lemma 6. *There exists an optimal schedule $\mathcal{S}_{OPT}$ having the property that, at any time, if there are ℓ active jobs, then the processor speed is at least $\sqrt[\alpha]{\ell/(\alpha-1)}$.*

Lemma 7. *For $c = \alpha - 1$, in $\mathcal{S}_{OPT}$ the n_1 jobs released at time t_0 are finished by time t_1 and the n_i jobs released during phase $i-1$ are finished by time t_i, for $i = 2, \ldots, k$.*

Lemma 8. *There holds $g(\mathcal{S}_{OPT}) \geq C_T^{1-\alpha}(1+\Phi)^{-1}(1+\Phi^{\alpha/(2\alpha-1)})^{(1-\alpha)}\sum_{i=1}^{k}n_i^{2-1/\alpha}+\sum_{i=1}^{k}T(n_i)$, where $\Phi = (1+\sqrt{5})/2$..*

Lemma 9. *For $c = \alpha - 1$, the competitive ratio of Phasebal is at most $(1+\Phi)(1+\Phi^{\alpha/(2\alpha-1)})^{(\alpha-1)}\frac{\alpha^\alpha}{(\alpha-1)^{\alpha-1}}\frac{5\alpha-2}{2\alpha-1}$.*

Proof. Using (6), (7) and Lemma 8, we can determine the ratio of the online cost to the optimal offline cost as in Lemma 5. The desired competitive ratio can then be derived using algebraic manipulations. The calculations are more involved than in the proof of Lemma 5. Details are given in the full version of the paper. $\square$

Theorem 2 now follows from Lemmas 5 and 9, observing that $\alpha_0 = (19+\sqrt{161})/10 \geq 2$ and that, for $\alpha > \alpha_0$, we have $\frac{4}{2\alpha-1}+\frac{4}{\alpha-1} < \frac{5\alpha-2}{2\alpha-1}$.

5 An Optimal Offline Algorithm for Unit-Size Jobs

We present a polynomial time algorithm for computing an optimal schedule, given a sequence of unit-size jobs that is known offline. Our algorithm is based on dynamic programming and constructs an optimal schedule for a given job sequence σ by computing optimal schedules for subsequences of σ. A schedule for σ can be viewed as a sequence of subschedules $S_1, S_2, \ldots, S_m$, where any S_j processes a subsequence of jobs $j_1, \ldots, j_k$ starting at time r_{j_1} such that $c_i > r_{i+1}$ for $i = j_1, \ldots, j_k - 1$ and $c_{j_k} \leq r_{j_k+1}$. In words, jobs j_1 to j_k are scheduled continuously without interruption such that the completion time of any job i is after the release time of job $i+1$ and the last job j_k is finished no later than the release time of job j_k+1. As we will prove in the next two lemmas, the optimal speeds in such subschedules S_j can be determined easily. For convenience, the lemmas are stated for a general number n of jobs that have to be scheduled in an interval $[t, t')$. The proofs are omitted.

Lemma 10. *Consider n jobs that have to be scheduled in time interval $[t, t')$ such that $r_1 = t$ and $r_n < t'$. Suppose that in an optimal schedule $c_i > r_{i+1}$, for $i = 1, \ldots, n-1$. If $t' - t \geq \sum_{i=1}^{n} \sqrt[\alpha]{(\alpha-1)/(n-i+1)}$, then the i-th job in the sequence is executed at speed $s_i = \sqrt[\alpha]{(n-i+1)/(\alpha-1)}$.*

Lemma 11. *Consider n jobs that have to be scheduled in time interval $[t, t')$ such that $r_1 = t$ and $r_n < t'$. Suppose that in an optimal schedule $c_i > r_{i+1}$, for $i = 1, \ldots, n-1$. If $t' - t < \sum_{i=1}^{n} \sqrt[\alpha]{(\alpha-1)/(n-i+1)}$, then the i-th job in the sequence is executed at speed $s_i = \sqrt[\alpha]{(n-i+1+c)/(\alpha-1)}$, where c is the unique value such that $\sum_{i=1}^{n} \sqrt[\alpha]{(\alpha-1)/(n-i+1+c)} = t' - t$.*

Of course, an optimal schedule for a given σ need not satisfy the condition that $c_i > r_{i+1}$, for $i = 1, \ldots, n-1$. In fact, this is the case if the speeds specified in Lemmas 10 and 11 do not give a feasible schedule, i.e. there exists an i such that $c_i = \sum_{j=1}^{i} t_j \leq r_{i+1}$, with $t_i = 1/s_i$ and s_i as specified in the lemmas. Obviously, this infeasibility is easy to check in linear time.

We are now ready to describe our optimal offline algorithm, a pseudo-code of which is presented in Figure 1. Given a jobs sequence consisting of n jobs, the algorithm constructs optimal schedules for subproblems of increasing size. Let $P[i, i+l]$ be the subproblem consisting of jobs i to $i+l$ assuming that the processing may start at time r_i and must be finished by time r_{i+l+1}, where $1 \leq i \leq n$ and $0 \leq l \leq n-i$. We define $r_{n+1} = \infty$. Let $C[i, i+l]$ be the cost of an optimal schedule for $P[i, i+l]$. We are eventually interested in $C[1, n]$. In an initialization phase, the algorithm starts by computing optimal schedules for $P[i, i]$ of length $l = 0$, see lines 1 to 3 of the pseudo-code. If $r_{i+1} - r_i \geq \sqrt[\alpha]{\alpha - 1}$, then Lemma 10 implies that the optimal speed for job i is equal to $\sqrt[\alpha]{1/(\alpha-1)}$. If $r_{i+1} - r_i < \sqrt[\alpha]{\alpha-1}$, then by Lemma 11 the optimal speed is $1/(r_{i+1} - r_i)$. Note that this value can be infinity if $r_{i+1} = r_i$. The calculation of $C[i, i]$ in line 3 will ensure that in this case an optimal schedule will not complete job i by r_{i+1}.

```
Algorithm Dynamic Programming
1. for $i := 1$ to $n$ do
2.     if $r_{i+1} - r_i \geq \sqrt[\alpha]{\alpha-1}$ then $S[i] := \sqrt[\alpha]{1/(\alpha-1)}$ else $S[i] := 1/(r_{i+1} - r_i)$;
3.     $C[i,i] := (S[i])^{\alpha-1} + 1/S[i]$;
4. for $l := 1$ to $n-1$ do
5.     for $i := 1$ to $n-l$ do
6.         $C[i,i+l] := \min_{i \leq j < i+l}\{C[i,j] + C[j+1,i+l]\}$;
7.         Compute an optimal schedule for $P[i,i+l]$ according to Lemmas 10 and 11
           assuming $c_j > r_{j+1}$ for $j = i, \ldots, i+l-1$ and let $s_i, \ldots, s_{i+l}$ be the
           computed speeds;
8.         if schedule is feasible then $C := \sum_{j=i}^{i+l} s_j^{\alpha-1} + \sum_{j=i}^{i+l}(i+l-j+1)/s_j$
           else $C := \infty$;
9.         if $C < C[i,i+l]$ then $C[i,i+l] := C$ and $S[j] := s_j$ for $j = i, \ldots, i+l$;
```

Fig. 1. The dynamic programming algorithm

After the initialization phase the algorithm considers subproblems $P[i, i+l]$ for increasing l. An optimal solution to $P[i, i+l]$ has the property that either (a) there exists an index j with $j < i+l$ such that $c_j \leq r_{j+1}$ or (b) $c_j > r_{j+1}$ for $j = i, \ldots, i+l-1$. In case (a) an optimal schedule for $P[i, i+l]$ is composed of optimal schedules for $P[i, j]$ and $P[j+1, i+l]$, which is reflected in line 6 of the pseudo-code. In case (b) we can compute optimal processing speeds according to Lemmas 10 and 11, checking if the speeds give indeed a feasible schedule. This is done in lines 7 and 8 of the algorithm. In a final step the algorithm checks if case (a) or (b) holds. The algorithm has a running time of $O(n^3 \log \rho)$, where ρ is the inverse of the desired precision. Note that in Lemma 11, c can be computed only approximately using binary search.

We briefly mention that we can use our dynamic programming approach to compute a schedule that minimizes the total flow time of jobs, given a fixed amount A of energy. Here we simply consider the minimization of a weighted objective function $g_\beta(\mathcal{S}) = \beta E(\mathcal{S}) + (1-\beta)F(\mathcal{S})$, where $0 < \beta < 1$. By suitably choosing β, we obtain an optimal schedule $\mathcal{S}_{OPT}$ for g_β with $E(\mathcal{S}_{OPT}) = A$. This schedule minimizes the flow time. Details can be found in the full version of the paper.

References

1. J. Augustine, S. Irani and C. Swamy. Optimal power-down strategies. *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, 530-539, 2004.
2. N. Bansal, T. Kimbrel and K. Pruhs. Dynamic speed scaling to manage energy and temperature. *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, 520–529, 2004.
3. N. Bansal and K. Pruhs. Speed scaling to manage temperature. *Proc. 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Springer LNCS 3404, 460–471, 2005.
4. G. Cornuéjols, G.L. Nemhauser and L.A. Wolsey. The uncapacitated facility location problem. In P. Mirchandani and R. Francis (eds.), *Discrete Location Theory*, 119–171, John Wiley & Sons, 1990.
5. D.R. Dooly, S.A. Goldman, and S.D. Scott. On-line analyis of the TCP acknowledgment delay problem. *Journal of the ACM*, 48:243–273, 2001.
6. A. Fabrikant, A. Luthra, E. Maneva, C.H. Papadimitriou and S. Shenker. On a network creation game. *Proc. 22nd Annual ACM Symposium on Principles of Distributed Computing*, 347–351, 2003.
7. S. Irani, S. Shukla and R. Gupta. Algorithms for power savings. *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 37–46, 2003.
8. A.R. Karlin, C. Kenyon and D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. *Proc. 33rd ACM Symposium on Theory of Computing*, 502–509, 2001.
9. P. Mirchandani and R. Francis (eds.). *Discrete Location Theory*. John Wiley & Sons, 1990.
10. K. Pruhs, P. Uthaisombut and G. Woeginger. Getting the best response for your erg. *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, Springer LNCS 3111, 15–25, 2004.

11. D.D. Sleator und R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202-208, 1985.
12. D. Shmoys, J. Wein and D.P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313-1331, 1995.
13. F. Yao, A. Demers and S. Shenker. A scheduling model for reduced CPU energy. *Proc. 36th Annual Symposium on Foundations of Computer Science*, 374–382, 1995.