Getting the Best Response for Your Erg

Kirk Pruhs [*] *Patchrawat Uthaisombut* [†] *Gerhard Woeginger* [‡]

Abstract

We consider the speed scaling problem of minimizing the average response time of a collection of dynamically released jobs subject to a constraint A on energy used. We propose an algorithmic approach in which an energy optimal schedule is computed for a huge A, and then the energy optimal schedule is maintained as A decreases. We show that this approach yields an efficient algorithm for equi-work jobs. We note that the energy optimal schedule has the surprising feature that the job speeds are not monotone functions of the available energy. We then explain why this algorithmic approach is problematic for arbitrary work jobs. Finally, we explain how to use the algorithm for equi-work jobs to obtain an algorithm for arbitrary work jobs that is $O(1)$-approximate with respect to average response time, given an additional factor of $(1+\epsilon)$ energy.

1 Introduction

Limiting power consumption has become a first-class architectural design constraint [13]. Speeding scaling, which involves dynamically changing the power over time, is a power management technique available in current microprocessors from AMD, Intel and Transmeta. The well known cube-root rule states speed is roughly proportional to the cube-root of the power, or equivalently, that the power is proportional to the speed cubed [7]. Here we are concerned about managing power to conserve energy, which is power integrated over time.

One natural question is then what speed scaling policy should the operating system use to set the speed at each point of time to best conserve energy. The speed setting policy is symbiotically related to the scheduling policy for determining which process to run. The operating system would like to both optimize some Quality of Service (QoS) measure that it provides to the applications, and to minimize the energy that it uses. In general these two objectives are in opposition, that is, the more energy that it is available, the better QoS that can be provided.

Theoretical algorithmic investigations into this problem were initiated in [16]. [16] formulated the problem as a scheduling problem. Further [16] assumed that each job had a deadline, and that the QoS measure was that each job completed by its deadline. One advantage of the deadline feasibility QoS measure is that the resulting optimization problem reduces to a single objective

[*]Computer Science Department. University of Pittsburgh. Pittsburgh PA 15260 USA. kirk@cs.pitt.edu. Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, and CCF-0448196, and CCF-0514058.

[†]Computer Science Department. University of Pittsburgh. Pittsburgh PA 15260 USA. utp@cs.pitt.edu.

[‡]Department of Mathematics and Computer Science. Eindhoven University of Technology. P.O. Box 513, 5600 MB Eindhoven, The Netherlands. gwoegi@win.tue.nl.

problem as the QoS considerations become a constraint instead of an objective. Recently, there has been a flurry of papers on speed scaling policies for scheduling problems with deadlines [16, 4, 3, 10, 12, 9, 8, 17, 11].

However, in general computational settings, most processes do not have natural deadlines associated with them. This is why operating systems like Unix and Microsoft Windows do not have deadline based schedulers. By far the most commonly used QoS measure in the computer systems literature is average response time. The response time (or flow time) of a process in a schedule is the amount of time between the time that the job is released and the time that the job is completed in that schedule. The average response time (or average flow time) is the average of the response times of all processes.

In this paper we initiate the study of the bi-criteria problem of minimizing average response time and minimizing energy usage. The simplest way to formalize a bi-criteria optimization problem is to bound one parameter and to optimize the other parameter. It seems most logical to us to bound the available energy, and then to minimize average response time. This is certainly most logical in a setting such as a laptop where energy is provided by a battery. This is the first theoretical algorithmic paper on speed scaling policies for a problem where the QoS measure does not involve deadlines.

In general, algorithm design and analysis are significantly more difficult in speed scaling problems than in the corresponding scheduling problem on a fixed speed processor. The reason for this is that in speed scaling problems there is no local constraint how much energy/power can used at each point in time. In particular, most of the local arguments used in the analysis of standard scheduling problems are inapplicable to speed scaling problems. As a representative example, consider the scheduling policy Shortest Remaining Processing Time (SRPT). SRPT is a preemptive scheduling policy that always runs the job with the least remaining work that has been released but not completed. SRPT is optimal with respect to average response time. In the analysis of SRPT, one proves the invariant that at all times, it is the case that the k jobs with maximum remaining work for SRPT have greater aggregate work than the k jobs with maximum remaining work for an arbitrary schedule. While SRPT remains the optimal scheduling policy with speed scaling, one can not prove the optimality of any speed scaling policy in such a local manner. The reason is that for any time, there may be schedules that have no remaining work at that time because they used most of their global energy resources just before this time. One of the reasons that all of the literature to date has considered scheduling problems with deadlines is that deadlines constrain how energy can reasonably allocated globally, making the resulting problems tractable to analyses.

In this paper we propose an algorithmic approach to speeding scaling problems that is potentially applicable to a wide range of speed scaling problems. The idea is to first compute an energy optimal schedule assuming that the energy bound A is huge, and then to observe the evolution of the energy optimal schedule as A decreases. In order for this approach to be effective, we need that our scheduling problem has three properties. Firstly, we obviously need that we can compute the optimal schedule when A is huge. For flow time problems, the optimal schedule when A is huge has each job running with the same large power, and each job finishing before the next job is released. Secondly, in order to track the evolution of the energy optimal schedule, this schedule must evolve in some sort of smooth manner as a function of A. It would seem that a lower bound to the time complexity of our approach is the number of structural changes in the optimal schedule as a function of A. So thirdly, in order to get an efficient algorithm, we need that the number of structural changes in the energy optimal schedule, as A decreases, should be say polynomially

bounded.

We first apply our approach to the case of equi-work jobs. We show that the problem of finding an energy optimal schedule can be expressed as a relatively simple convex program. The Karush-Kuhn-Tucker (KKT) conditions then give necessary structural conditions that an energy optimal schedule must satisfy. The KKT conditions seem to be a useful tool in speed scaling problems (see also [4]). These structural conditions allow us to track the energy optimal schedule as A decreases. We show that the energy optimal schedule changes continuously as a function of the energy bound A. Although we discover one surprising feature of energy optimal schedules. Intuitively the less energy available, the slower the jobs should be run in the energy optimal schedule. But we show that in fact this intuition is not always correct. That is, as energy decreases, some jobs can actually be run at a higher speed in the optimal schedule. So none of the obvious properties (speed, energy used, etc.) of a job are monotone functions of energy. We also show that the number of structural changes in the energy optimal schedule as A decreases is linearly bounded. Thus we essentially obtain an $O(n^2 \log L)$ time algorithm to find all energy optimal schedules, as well as the energy range where each schedule is optimal. Here L is the range of possible energies divided by the precision that we desire.

In contrast, we show that applying our algorithmic approach to arbitrary work jobs is problematic. More precisely, we show that the energy optimal schedule can be a discontinuous function of the energy bound A. However, we are able to obtain a resource augmentation result. We explain how to use the algorithm for equi-work jobs to obtain an algorithm for arbitrary work jobs that is $O(1)$-approximate with respect to average response time, given an additional factor of $(1 + \epsilon)$ energy.

Because our problems are non-linear, we generally get solutions where the variables are not only irrational, but don't appear to have any simple closed-form representation. In this paper, we will brush this finite precision issue under the carpet. Handling this issue is rather orthogonal to the combinatorial issues in which we are interested.

One of our students, Aleksandar Ivetic, implemented a variation of our algorithm in Mathematica, and created a GUI in Java. The GUI lets the user enter an arbitrary job instance. The GUI also has a slider that lets the user change the available energy, and view the resulting optimal schedule. By moving the slider, the user essentially gets a movie of the evolving optimal schedule. The software, and additional information, can be found at `http://www.cs.pitt.edu/~utp/energy`.

1.1 Related Results

All prior and contemporaneous theoretical speed scaling results have QoS measures involving deadlines. [16] considers where the QoS measure is deadline feasibility and the objective is to minimize energy. They give a greedy algorithm that is optimal for the offline setting. They propose two online algorithms Optimal Available (OA) and Average Rate (AVR). They show that AVR is $O(1)$-competitive. In [3] it is shown that OA is $O(1)$-competitive. [3] introduces the problem of minimizing temperature subject to deadline feasibility constraints under the assumption that the processor cools according to Newton's law. [3] shows that in principle the temperature optimal schedule can be computed by the Ellipsoid Method. [3] introduces another online algorithm BKP. [4] shows that BKP is $O(1)$-competitive with respect to temperature, independent of the cooling parameter in Newton's law. [4] shows that as a consequence of this, BKP is $O(1)$-competitive with respect to energy (this was originally shown in [3]). [3] shows that BKP is optimally competitive

with respect to maximum power.

A naive implementation of the offline algorithm in [16] runs in time $O(n^3)$. This can be improved to $O(n^2)$ if the intervals have a tree structure [12]. For jobs with a fixed priority, [17] shows that it is NP-hard to compute an minimum energy schedule. They also give a fully polynomial time approximation scheme for the problem. [11] gives a polynomial time algorithm for the case of a processor with discrete speeds. Results for problems that add profit maximization, and precedence constraints, to deadline scheduling can be found in [9, 8].

Deadline scheduling with sleep states is considered in [10, 2]. Deadline scheduling on multiple machines with precedence constraints is considered in [15].

2 Definitions, Notation, and Preliminaries

The input consists of n jobs, referenced by the integers from 1 to n. Job i arrives at its release time r_i. For now we assume that no release dates are equal, and that the jobs are ordered so that $r_1 < r_2 < ... < r_n$. We'll later explain how to generalize to the case that jobs can have identical release dates. Job i has a positive work requirement w_i. The jobs will be processed on one machine. A *schedule* specifies, for each time, the job that is run, and the speed at which that job is run. We assume preemption, that is, the execution of a job can be suspended and later restarted from the point of suspension. A job can not be run before its release date. If a job i is run at speed s for t units of time, then $s \cdot t$ units of work are completed from i. The job i is completed at the time C_i when all of its work has been completed. Let $F_i = C_i - r_i$ be the response time (flow time) of job i, and $F = \sum_{i=1}^{n}(C_i - r_i)$ be the total response time (total flow time). There is a bound A on the amount of available energy. If a job i is run at speed s then we assume the power, energy used per unit time, is s^α. We assume throughout the paper that $\alpha \geq 2$. A schedule is *feasible at energy level* A if it completes all jobs, and the total amount of energy used is no more than A. We say that a feasible schedule is *optimal for energy level* A if it has minimum total response time (or equivalently, minimum average response time) among all feasible schedules at energy level A. There is always an optimal schedule that runs each job at a fixed speed. This was noted in [16], and immediately follows from the convexity of the speed to power function. We will therefore restrict our attention to schedules where each job is run at a fixed speed. For a job i in a particular schedule, let s_i denote its speed, $x_i = \frac{w_i}{s_i}$ its processing time, $p_i = s_i^\alpha$ its power, and $e_i = x_i p_i = s_i^{\alpha-1} w_i$ its energy consumption. Let $E = \sum_{i=1}^{n} e_i$ be the total energy used. To refer to a particular schedule S, when the schedule in question is not clear from context, we append (S) to our notation. So for example, $F(S)$ is the total response time for schedule S, and $e_i(S)$ is the energy used by job i in schedule S.

In the equi-work case, we scale our unit of work so that each job has unit work. The scheduling policy First-Come-First-Served (FCFS) non-preemptively schedules jobs according to the order that they arrive. In the equi-work case, it is easy to see that SRPT and FCFS behave identically. and are optimal scheduling policies for equi-work jobs. Thus, without loss of generality, we may only consider schedules where $C_1 < C_2 < ... < C_n$, and no jobs are preempted.

3 Unit Work Jobs

In this section we consider the case that jobs have unit work.

3.1 Convex Programming Formulation

The problem of minimizing total response time for unit work jobs subject to a bound A on the total energy energy used can be expressed as the following convex program:

$$\min \sum_{i=1}^{n} C_i \tag{1}$$

$$\sum_{i=1}^{n} \frac{1}{x_i^{\alpha-1}} \leq A \tag{2}$$

$$C_{i-1} + x_i \leq C_i \qquad i = 2, \ldots, n \tag{3}$$

$$r_i + x_i \leq C_i \qquad i = 1, \ldots, n \tag{4}$$

$$-x_i \leq -\frac{1}{A^{1/(\alpha-1)}} \qquad i = 1, \ldots, n \tag{5}$$

The objective function is the total completion time, which is equivalent to the total response time. Each summand in line 2 is the amount of energy used by job i. The inequalities in lines 3 and 4 express the fact that, due to the FCFS scheduling policy, the starting time for each job i is the latter of the release time of job i and the completion time of job $i-1$. The inequality in line 5 holds since $s_i = \frac{1}{x_i}$ and $s_i^{\alpha-1}$ is the energy used by a unit work job. This constraint insures the positivity of the x_i's. Further, since $\alpha \geq 2$, the equation in line 2 implies that each x_i must satisfy $\frac{1}{x_i} < A^{1/(\alpha-1)}$. Note that equation 2 by itself does not ensure the positivity of x_i's.

Since the objective and all constraints other than the one in line 2 are linear, to verify that this is a convex program one needs only check that the inequality in line 2 defines a convex region. From equation 5, we only need to consider the subregion where x_i's are positive. Let $f(\vec{x}) = \sum_{i=1}^{n} \frac{1}{x_i^{\alpha-1}}$. Let $\vec{a}$ be a vector such that $f(\vec{a}) \leq A$ and $a_i > 0$ for $i = 1, ..., n$. Let $\vec{b}$ be a vector such that $f(\vec{b}) \leq A$ and $b_i > 0$ for $i = 1, ..., n$. Then it is sufficient to show that $f((\vec{a}+\vec{b})/2) \leq (f(\vec{a}) + f(\vec{b}))/2$, which is equivalent to $\sum_{i=1}^{n} \frac{1}{((a_i+b_i)/2)^{\alpha-1}} \leq (\sum_{i=1}^{n} \frac{1}{a_i^{\alpha-1}} + \sum_{i=1}^{n} \frac{1}{a_i^{\alpha-1}})/2$. To show this, we will show that $\frac{1}{((a_i+b_i)/2)^{\alpha-1}} \leq (\frac{1}{a_i^{\alpha-1}} + \frac{1}{a_i^{\alpha-1}})/2$ for $i = 1, ..., n$. By straightforward algebraic manipulation, this is equivalent to $2^{\alpha} \leq (1+R)^{\alpha-1} + (1+1/R)^{\alpha-1}$ where $R = \max(b_i/a_i, a_i/b_i)$. The inequality holds because (1) when $R = 1$, the inequality becomes an equality, (2) $R \geq 1 \geq 1/R$, and (3) the differentiation of the right side with respect to R is positive.

All the constraints can be easily transformed into the form $f_i(x) \leq 0$. This will allow us to easily apply the KKT conditions discussed in the next subsection. The gradients of the constraint and objective functions in this convex program can be computed easily. Such convex programs can be solved to arbitrary precision, in polynomial time, by the Ellipsoid algorithm [14]. Our goal is to find a simpler, more efficient, and more combinatorial algorithm.

3.2 KKT Conditions

The well-known Karush-Kuhn-Tucker (KKT) conditions provide a necessary and sufficient condition to establish the optimality of a particular feasible solution to certain convex programs [5]. Consider

a convex program

$$\min f_0(x) \tag{6}$$

$$f_i(x) \leq 0 \qquad i = 1, \ldots, n \tag{7}$$

Assume that this program is strictly feasible, that is, there is some point x where where $f_i(x) < 0$ for $i = 1, \ldots, n$. Assume that the f_i are all differentiable. Let λ_i, $i = 1, \ldots, n$ be a dual variable (Lagrangian multiplier) associated with the function $f_i(x)$. Then the necessary and sufficient KKT conditions for solutions x and λ to be primal and dual feasible are:

$$f_i(x) \leq 0 \qquad i = 1, \ldots, n \tag{8}$$

$$\lambda_i \geq 0 \qquad i = 1, \ldots, n \tag{9}$$

$$\lambda_i f_i(x) = 0 \qquad i = 1, \ldots, n \tag{10}$$

$$\nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) = 0 \tag{11}$$

Here $\nabla f_i(x)$ is the gradient of $f_i(x)$. Condition 10 is called complementary slackness.

We now apply the KKT conditions to our convex program to deduce necessary conditions for a feasible solution to be optimal.

Lemma 1. *If a primary feasible solution to our convex program is optimal, then:*

- *A total energy of A is used.*
- *For each job i, with $C_i < r_{i+1}$, it is the case that $p_i = p_n$.*
- *For each job i, with $C_i > r_{i+1}$, it is the case that $p_i = p_{i+1} + p_n$.*
- *For each job i, with $C_i = r_{i+1}$, it is the case that $p_n \leq p_i \leq p_{i+1} + p_n$.*

Proof. We apply the KKT conditions. Note that our convex program is strictly feasible because running the jobs infinitely slowly results in zero energy used. Thus line 2 is strictly feasible. We associate a dual variable σ with equation 2, a dual variable β_i, $2 \leq i \leq n$, with equation i in line 3, a dual variable γ_i, $1 \leq i \leq n$, with equation i in line 4, and a dual variable δ_i, $1 \leq i \leq n$, with equation i in line 5. We now consider equation 11 of the KKT conditions. Looking at the component of equation 11 corresponding to the variable C_1, we get that

$$1 + \beta_2 - \gamma_1 = 0 \tag{12}$$

Looking at the component of equation 11 corresponding to the variable C_i, $2 \leq i \leq n-1$, we get that

$$1 - \beta_i + \beta_{i+1} - \gamma_i = 0 \quad \text{or equivalently} \quad \beta_i + \gamma_i = \beta_{i+1} + 1 \tag{13}$$

Looking at the component of equation 11 corresponding to the variable C_n, we get that

$$1 - \beta_n - \gamma_n = 0 \tag{14}$$

Looking at the component of equation 11 corresponding to the variable x_1 we get that

$$\sigma \frac{1-\alpha}{x_1^{\alpha}} + \gamma_1 - \delta_1 = 0 \tag{15}$$

Looking at the component of equation 11 corresponding to the variable x_i, $2 \leq i \leq n$, we get that

$$\sigma \frac{1-\alpha}{x_i^\alpha} + \beta_i + \gamma_i - \delta_i = 0 \tag{16}$$

As mentioned before, none of the equations in line 5 can ever be satisfied with equality in a feasible solution. Thus, by the complementary slackness, we know that each $\delta_i = 0$. Therefore from equation 16, we know that

$$\beta_i + \gamma_i = \sigma \frac{\alpha - 1}{x_i^\alpha} = \sigma(\alpha - 1)p_i \tag{17}$$

So that $i = 1$ is not a special case, let us introduce a new variable β_1 and set it to zero. Note that from equation 14, $\beta_n + \gamma_n = 1$, and hence $p_n = \frac{1}{\sigma(\alpha-1)}$. Therefore it must be the case that $\sigma > 0$. By complementary slackness, equality 2 must be tight. Or equivalently, A units of energy must be used.

We now show that these relationships between the β_i and γ_i dual variables imply the last three items in the statement of the lemma. From equalities 13 and 9, β_i and γ_i cannot be both 0. By complementary slackness, inequalities 3 and 4 cannot be strict inequalities at the same time. Consider a job $i = 1, ..., n-1$, and the following three cases that might arise:

- Consider the case that $C_i < r_{i+1}$. Thus, $C_i + x_{i+1} < r_{i+1} + x_{i+1} = C_{i+1}$. By complementary slackness, it must be the case that $\beta_{i+1} = 0$. By equalities 17 and 13, $\sigma(\alpha-1)p_i = \beta_i + \gamma_i = \beta_{i+1} + 1 = 1$. Therefore $p_i = \frac{1}{\sigma(\alpha-1)}$, which is equal to p_n.

- Consider the case that $r_{i+1} < C_i$. Thus, $r_{i+1} + x_{i+1} < C_i + x_{i+1} = C_{i+1}$. By complementary slackness, it must be the case that $\gamma_{i+1} = 0$. By equalities 17 and 13,

  $$\sigma(\alpha-1)p_i = \beta_i + \gamma_i = \beta_{i+1} + 1 = \beta_{i+1} + \gamma_{i+1} + 1 = \sigma(\alpha-1)p_{i+1} + 1.$$

  Therefore,

  $$p_i = p_{i+1} + \frac{1}{\sigma(\alpha-1)} = p_{i+1} + p_n.$$

- Finally, consider the case that $r_{i+1} = C_i$. Thus, $r_{i+1} + x_{i+1} = C_i + x_{i+1} = C_{i+1}$. By equalities 17 and 13,

  $$\sigma(\alpha-1)p_i \;=\; \beta_i + \gamma_i \;=\; \beta_{i+1} + 1 \;\leq\; \beta_{i+1} + \gamma_{i+1} + 1 \;=\; \sigma(\alpha-1)p_{i+1} + 1.$$

  Or equivalently,

  $$p_n \;\leq\; p_i \;=\; \frac{\beta_{i+1}}{\sigma(\alpha-1)} + \frac{1}{\sigma(\alpha-1)} \;\leq\; p_{i+1} + p_n.$$

□

Note that Lemma 1 does not explain how to find an optimal schedule for three reasons. Firstly, it does not not specify the correct value of p_n. Secondly, it does not directly specify the power for a job that completes exactly when the next job is released. And thirdly, and most significantly, it does not specify the relationship is between each C_i and r_{i+1} in the optimal schedule (whether $C_i < r_{i+1}$, $C_i = r_{i+1}$, or $C_i > r_{i+1}$). This relationship is crucial, and leads us to define what we call configuration curves.

3.3 Configuration Curves

We now introduce some definitions that are necessary before continuing our discussion. A *configuration* ϕ maps each job i, $1 \leq i \leq n-1$, to a relational symbol in the set $\{<, =, >\}$. A schedule is in configuration ϕ if C_i $\phi(i)$ r_{i+1} for all jobs i. So for example if $\phi(i)$ is $<$, then $C_i < r_{i+1}$. Define a *configuration curve* $M_\phi(A)$ to be a function that takes as input an energy bound A and outputs the optimal schedule (or polymorphically the total response time for this schedule), among those schedules in configuration ϕ that use at most A units of energy. Note that $M_\phi(A)$ may not be defined for all A. Let Φ be the set of all configurations. Define $M(A)$ as

$$M(A) = \min_{\phi \in \Phi} M_\phi(A).$$

Our goal is to compute $M(A)$, which is the lower envelope of the exponentially many configuration curves. The problem is non-trivial because (1) it is non-trivial to determine the optimal configuration for a given amount of energy, and (2) even if the optimal configuration is known, it is non-trivial to determine how the energy should be distributed to the n jobs to minimize total response time.

As an example, consider an instance with 3 jobs with release times 0, 1 and 3, and $\alpha = 3$. There are five configuration curves that are part of $M(A)$, namely, $<<$, $=<$, $><$, $>=$, and $>>$. The superposition of all five configuration curves that make up $M(A)$ are show in Figure 1. Note that configuration curves are in general only defined over a subrange of possible energies. The vertical lines indicate the energy levels where the optimal schedule changes configuration. The corresponding configurations and schedules are shown in Figure 2. Variable d in the top schedule is any constant less than 1. It indicates that the completion times of the 3 jobs are related. A similar relationship is shown in the bottom schedule.

Note that in Figure 2 the job released at time 1 is run at faster speeds as energy decreases from 1.282 to 1.147. This seems to go against the intuition that the less energy available, the slower the jobs should be run in the energy optimal schedule. This intuition is incomplete. There is another factor that influences job speeds. We now explain this factor. Consider a situation when jobs run back-to-back. If we can shorten the execution time of the last job in the chain by a little bit, then the flow time of that job will decrease a little bit. However, if we can shorten the execution time of the first job in the chain by the same amount, then the flow time of each job in the whole chain will decrease. So, jobs that run earlier in a chain is, in some sense, more important than jobs that run later in the chain. Thus, relatively more energy should be invested in jobs that run earlier in the chain. This factor plays a role in this situation.

Thus, there are two factors that affects the power of job interacting with each other, (1) the total amount of energy, and (2) the relative importance of jobs. In our situation in Figure 2, the importance of the second job become greater relative to the importance of the first job as the energy decreases from 1.282 to 1.147. In particular, the ratio of the powers for the first two jobs ($p_1 : p_2$) is changing from $2 : 1$ to $3 : 2$. In this particular situation, factor (2) has more influence than factor (1). Thus, the net effect is that the power of the second job increases even though the total amount of energy is decreasing.

3.4 Basic Algorithmic Strategy

We are now ready to describe our basic algorithmic strategy. Intuitively, we trace out the lower envelope $M(A)$ of the configuration curves. We start with A being large, and then continuously

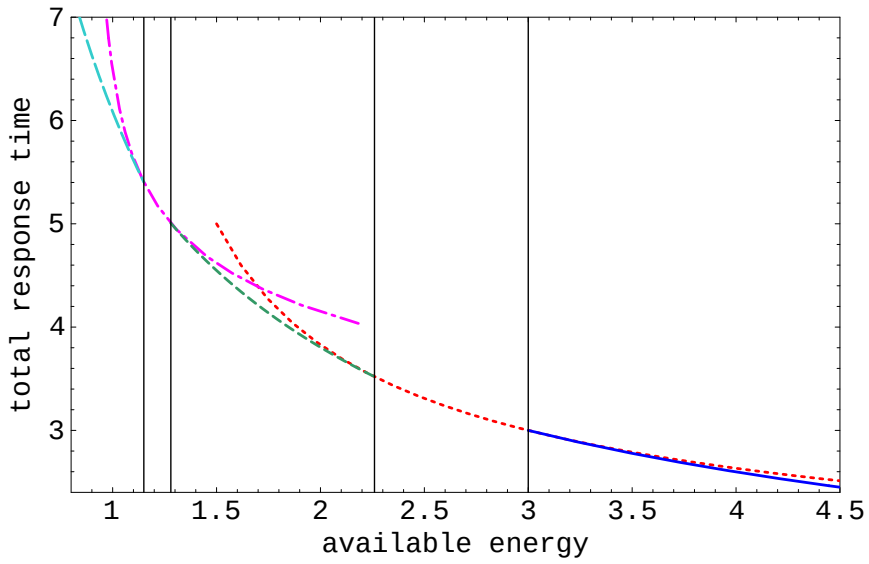

Figure 1: The superposition of the five configuration curves that make up $M(A)$.

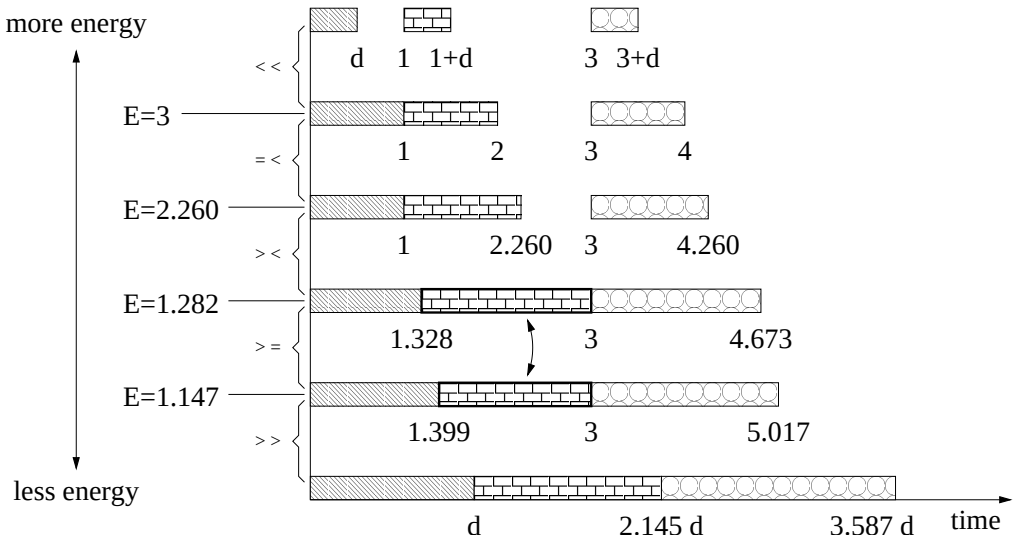

Figure 2: Optimal schedules at different amounts of available energy

decrease A. When A is sufficiently large then it is easy to see that in the optimal schedule $C_i < r_{i+1}$ for all i, and that all jobs are run at the same power. We have to address three issues.

- In section 3.5, we show that the optimal schedule is unique for each energy level.
- In section 3.6, we explain how to compute $M_\phi(A)$ for any given configuration ϕ.
- In section 3.7, we explain how to recognize when the optimal configuration changes from a configuration ϕ to a configuration ϕ'.

If we can accomplish these goals then we could trace out $M(A)$ by continuously decreasing A. Actually, rather than working with the energy bound A, it will prove more convenient to work with the power p_n of job n. We will eventually show in Lemma 6 that moving continuously from a high p_n to a low p_n is equivalent to continuously moving from a high A to a low A. In section 3.8 we explain how to make this algorithm efficient by discretely changing A and p_n using binary search.

3.5 The Optimal Schedule for Each Energy Level is Unique

We show that the optimal schedule is unique for each energy level. Thus when two configuration curves on the lower envelope intersect, we know that the underlying schedules are identical.

Lemma 2. *For each energy bound A, the optimal schedule is unique.*

Proof. Assume to reach a contradiction that you have two different optimal schedules S and T. Consider a third schedule U where for all jobs i, $x_i(U) = (x_i(S) + x_i(T))/2$. We now claim that $F(U) \leq (F(S) + F(T))/2 = F(S) = F(T)$ and $E(U) < (E(S) + E(T))/2$. From this it follows that neither S or T is optimal since one can get a better schedule by reinvesting $A - E(U)$ energy into job n in U to get a schedule with better response time than $F(U)$. This contradicts the optimality of S and T.

To see that $F(U) \leq (F(S) + F(T))/2 = F(S) = F(T)$ consider a particular job b. Then there exists some job a such that $C_b(U) = r_a + \sum_{i=a}^{b} x_i(U)$. Therefore by the definition of U, $C_b(U) = r_a + \sum_{i=a}^{b} (x_i(S) + x_i(T))/2$. But in S it must be the case that $C_b(S) \geq r_a + \sum_{i=a}^{b} x_i(S)$ since S must process jobs a through b between time r_a and $C_b(S)$. Similarly, $C_b(T) \geq r_a + \sum_{i=a}^{b} x_i(T)$. By averaging these two equations, $(C_b(S) + C_b(T))/2 \geq r_a + \sum_{i=a}^{b} (x_i(S) + x_i(T))/2$. We know the right-hand side of this inequality is exactly $C_b(U)$. Hence, $(C_b(S) + C_b(T))/2 \geq C_b(U)$. Since b was arbitrarily chosen, it follows by summing that $F(U) \leq (F(S) + F(T))/2$.

Note that the function $f(x) = \frac{1}{x^{\alpha-1}}$ is a convex function when $\alpha > 1$, and $f(\frac{a+b}{2}) < (f(a) + f(b))/2$. It then immediately follows that $E(U) < (E(S) + E(T))/2$ on a job by job basis since $e_i(U) = \frac{1}{((x_i(S)+x_i(T))/2)^{\alpha-1}}$, and $(e_i(S) + e_i(T))/2 = (\frac{1}{x_i(S)^{\alpha-1}} + \frac{1}{x_i(T)^{\alpha-1}})/2$. □

3.6 How to Compute $M_\phi(A)$ for a given configuration ϕ

We consider the problem of, given an energy A, and a fixed configuration ϕ, finding the optimal schedule among those schedules with configuration ϕ and energy at most A. Actually, we restate this problem as: given a power p_n, and a fixed configuration ϕ, finding the optimal schedule among those schedules with configuration ϕ and power p_n on job n.

We define a *group* as a maximal substring of jobs where $\phi(i)$ is $>$. More precisely $a, a+1, \ldots, b$ is a group if

- $a = 1$, or $\phi(a-1)$ is not $>$, and
- for $i = a, ..., b-1$ it is the case that $\phi(i)$ is $>$, and
- $b = n$, or $\phi(b)$ is not $>$.

A group is *open* if $\phi(b)$ is $<$ or $b = n$, and it is *closed* if $\phi(b)$ is $=$. Lemma 3 states how to compute the powers of the jobs in a group given the power of the last job of the group.

Lemma 3. *If $a, a+1, \ldots, b$ are jobs in a group in the optimal schedule for energy level A, then $p_i = p_b + (b-i)p_n$ for $i = a, ..., b$.*

Proof. This is an immediate application of the third item in Lemma 1. □

It should now be clear how to compute the powers of the jobs $a, a+1, \ldots, b$ in an open group. From Lemma 1 the power of the last job in the group is p_n, and the powers of the earlier jobs in the group are given by Lemma 3. To compute the powers of the jobs in a closed group is a bit more involved. Lemma 4 establishes a relationship between the power of the jobs in a closed group and the length of the time period when the jobs in this group are run.

Lemma 4. *If $a, a+1, \ldots, b$ are jobs in a closed group in the optimal schedule for energy level A, then $\sum_{i=a}^{b} \frac{1}{(p_b+(b-i)p_n)^{1/\alpha}} = r_{b+1} - r_a$.*

Proof. From the definition of closed group, jobs $a, a+1, ..., b$ run back-to-back, job a starts at time r_a, and job b completes at time r_{b+1}. Thus,

$$r_{b+1} - r_a = \sum_{i=a}^{b} x_i = \sum_{i=a}^{b} \frac{1}{s_i} = \sum_{i=a}^{b} \frac{1}{(p_b + (b-i)p_n)^{1/\alpha}}$$

where the last equality follows from Lemma 3, □

Lemma 4 gives an implicit definition of p_b as a function of p_n. However, it does not appear possible express p_b as a simple closed-form function of p_n. We next prove in Lemma 5 that $\sum_{i=a}^{b} \frac{1}{(p_b+(b-i)p_n)^{1/\alpha}}$ is strictly increasing as a function of p_b. Therefore, one can determine p_b from p_n by binary search on p_b. We can then compute the powers of other jobs in this closed group using Lemma 3.

Lemma 5. *When p_n is fixed, $\sum_{i=a}^{b} \frac{1}{(p_b+(b-i)p_n)^{1/\alpha}}$ decreases as p_b increases.*

Proof. For $i = a, ..., b$, as p_b increases, $(p_b + (b-i)p_n)^{1/\alpha}$ increases. Thus, $1/(p_b + (b-i)p_n)^{1/\alpha}$ decreases, and so does $\sum_{i=a}^{b} \frac{1}{(p_b+(b-i)p_n)^{1/\alpha}}$. □

Finally, we show that continuously decreasing p_n is equivalent to continuously decreasing the energy bound A in the sense that they will trace out the same schedules, albeit at a different rate.

Lemma 6. *In the optimal schedules, the energy bound A is a strictly increasing continuous function of p_n, and similarly, p_n is a strictly increasing continuous function of A.*

Proof. We first prove that there is a bijection between p_n and A in the optimal schedules, as well as in the optimal schedules restricted to a particular configuration. The fact that a fixed p_n is mapped into a unique energy should be obvious from our development to date. That two different p_n's can not map to optimal schedules with the same energy follows from Lemma 2.

Since the function from p_n to A is obviously continuous, it then follows that the function from p_n to A is either strictly increasing or strictly decreasing. The fact that function is strictly increasing then follows from looking at the extreme points. If A is very large, then the optimal configuration is all $<$, and p_n is large. If A is very small, then the optimal configuration is all $>$, and p_n is small. □

The results of Lemmas 3 and 4 implies that if we know the configuration of the optimal schedule that uses energy A, then we can compute the actual schedule. Since there are $\Theta(3^n)$ possible configurations, then a simple algorithmic approach to compute the optimal schedule is to (1) enumerate all these configurations, (2) compute the schedule that is uses energy A for each of these configurations, and (3) select the one with the smallest total flow time. However, some of these schedules are not feasible, and need to be removed from consideration. These schedules can be detected using the observations in Section 3.7. Note that the fact that some of these schedules are not feasible does not contradict the results of Lemmas 3 and 4. These lemmas assume that the configuration corresponds to the optimal schedule. If the configuration does not correspond to the optimal schedule, then the schedule obtained may be infeasible. This algorithm runs in time $\Theta(\text{poly}(n)3^n)$ as there are $\Theta(3^n)$ possible configurations. This algorithm is different from the one described in Section 3.4.

3.7 How to Recognize When the Optimal Configuration Changes

In the algorithm described in Section 3.4, the algorithm gradually decreases the amount of energy A (or equivalently, the power p_n of job n). Eventually, the configuration of the optimal schedule will change from a configuration ϕ to a configuration ϕ'. We now explain how to recognize this. Given a configuration ϕ that is no longer optimal for energy level A, the formulas for computing the powers of the jobs in Lemmas 3 and 4 will yield a schedule whose configuration is not equal to ϕ. There are 2 ways that this could happen.

- One of the $<$ constraints is violated.
- There is a last job b in some closed group with $p_b > p_{b+1} + p_n$.

In a non-degenerate case, the configuration of the schedule obtained from Lemmas 3 and 4 will be different from ϕ by exactly one constraint $\phi(i)$. The next configuration ϕ' is obtained from ϕ by changing $\phi(i)$ as follows. If $\phi(i)$ is $<$, it should be changed to $=$. If $\phi(i)$ is $=$, it should be changed to $>$. In a degenerate case, all violated $\phi(i)$'s need to be changed.

3.8 Implementation Details and Time Complexity

To construct an efficient implementation of the algorithm described in Section 3.4 we need to change p_n in a discrete fashion. This can be accomplished using binary search to find the next value for p_n when the optimal configuration changes. Suppose L is the range of possible values of p_n divided by our desired accuracy. Then binary search needs to check $O(\log L)$ values of p_n to determine the next

value for p_n when the optimal configuration changes. The condition for determining whether the current configuration is optimal for a particular p_n, described in the Section 3.7, can be computed in linear time. Thus in time $O(n \log L)$ we can find the next configuration curve on the lower envelope. There are only $2n - 1$ configuration curves on the lower envelope as the only possible transitions are from $<$ to $=$, or from $=$ to $>$. Thus we get a running time of $O(n^2 \log L)$.

If there are jobs with equal release dates then the only change that is required is that in the initial configuration all jobs with equal release dates are in one open group with speeds given by Lemma 3.

3.9 Mathematica and Java Animation program

The animation program mentioned in the introduction is a variation of our algorithm. To allow quick response time for animation, the schedules for energy levels within the given range for the given accuracy are precomputed. To simplify the implementation, when the program computes the schedule for a particular energy level A, it does not trace to lower envelop $M(.)$ from high energy to low energy. Instead it begins with a configuration ϕ where each $\phi(i)$ is $<$, then computes the optimal schedule in this configuration at energy level A. If the resulting schedule violates the configuration ϕ, it updates the configuration ϕ as described in the previous section, and recomputes the optimal schedule for the new configuration. The program repeats this process until the resulting schedule matches ϕ.

4 Arbitrary Work Jobs

We now consider the case of arbitrary work jobs. We first show that applying our algorithmic approach in this case is problematic as the energy optimal schedule is not a smooth function of of the energy bound A. In particular, it is sufficient to consider an instance with two jobs where the job with the earlier release date has more work than the job with the later release date. Then consider what happens as the energy bound A decreases. First C_1 becomes equal to r_2. Then $C_1 > r_2$, but SRPT still gives priority to job 1 at time r_2, and $C_1 < C_2$. But eventually SRPT will give priority to job 2 at time r_2, and job 1 is preempted by job 2. Thus, $C_2 < C_1$. When this happens the value of $\max(C_1, C_2)$ increases discontinuously. See Figure 3. This discontinuous change can have a ripple effect as it can disturb any schedule of other jobs lying between the previous value of $\max(C_1, C_2)$ and the new value of $\max(C_1, C_2)$.

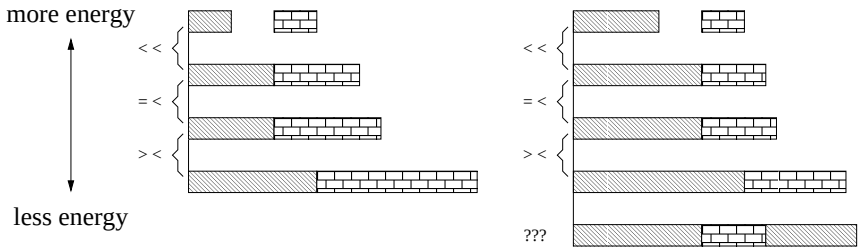

Figure 3: Optimal configurations at different amounts of available energy when jobs have unit work (left) and arbitrary amount of work (right)

To show that the makespan of the schedule with 2 jobs changes discontinuously when the jobs switch priority, we give the analysis of the situation. Let S be the optimal schedule under the restriction that job 1 is given a higher priority than job 2. Let S' be the optimal schedule under the restriction that job 2 is given a higher priority than job 1. It can be shown through an argument similar to that in Lemma 1 that $p_1(S) = 2p_2(S)$ and $p_2(S') = 2p_1(S')$. Or equivalently,

$$\begin{aligned} x_2(S) &= x_1(S)2^{1/\alpha}\frac{w_2}{w_1} \quad \text{and} \\ x_2(S') &= x_1(S')\frac{1}{2^{1/\alpha}}\frac{w_2}{w_1} \end{aligned} \tag{18}$$

Suppose A units of energy is available. Then,

$$\begin{aligned} A &= p_1(S)x_1(S) + p_2(S)x_2(S) = \frac{1}{x_1(S)^{\alpha-1}}(w_1^\alpha + \frac{1}{2^{1-1/\alpha}}w_2w_1^{\alpha-1}) \quad \text{and} \\ A &= p_1(S')x_1(S') + p_2(S')x_2(S') = \frac{1}{x_1(S')^{\alpha-1}}(w_1^\alpha + 2^{1-1/\alpha}w_2w_1^{\alpha-1}). \end{aligned}$$

$$\text{Thus,} \qquad x_1(S') = x_1(S)\left(\frac{w_1^\alpha + 2^{1-1/\alpha}w_2w_1^{\alpha-1}}{w_1^\alpha + \frac{1}{2^{1-1/\alpha}}w_2w_1^{\alpha-1}}\right)^{\frac{1}{\alpha-1}} \tag{19}$$

The optimal schedule is either S or S' and is equal to the one with the smaller flow time. $F(S)$ is smaller if A is large, and $F(S')$ is smaller if A is smaller.

$$\begin{aligned} F(S) &= F_1(S) + F_2(S) = (r_1 + x_1(S)) + (r_1 + x_1(S) + x_2(S)) \\ F(S') &= F_1(S') + F_2(S') = (r_1 + x_1(S') + x_2(S')) + (r_2 + x_2(S')) \end{aligned}$$

The switch over occurs when $F(S) = F(S')$, or equivalent when

$$r_1 + 2x_1(S) + x_2(S) = r_2 + x_1(S') + 2x_2(S'). \tag{20}$$

When this occurs, the makespan switches from $C_{\max}(S) = r_1 + x_1(S) + x_2(S)$ to $C_{\max}(S') = r_1 + x_1(S') + x_2(S')$. Now we argue that in general $C_{\max}(S) \neq C_{\max}(S')$. Suppose $C_{\max}(S) = C_{\max}(S')$, then from equality 20, this implies that $r_2 + x_2(S') = r_1 + x_1(S)$. However, from equalities 18 and 19, this is not true in general.

Note that we don't have this problem when jobs have unit work because SRPT behaves like FCFS. In particular, job priorities are fixed according to their arrival times. Thus, jobs never switch priorities as the energy decreases. For two unit-size jobs, the optimal schedule is always S, and never S'.

We can use the algorithm for equi-work jobs to obtain an algorithm for arbitrary work jobs that is $O(1)$-approximate with respect to average response time, given an additional factor of $(1 + \epsilon)$ energy. The *fractional response time* is the integral over times t, of the sum over all jobs j released before time t, of the fraction of the work of job j not completed by time t. So if $\frac{1}{3}$ of the work of job j is completed by time t, then job j contributes $\frac{2}{3}$ towards the sum at time t. The fractional response time is a lower bound to the total response time, which is the integral over all times t of the number of jobs released, but not completed, by time t. By discretizing time, we can write a convex program for the optimal fractional response time schedule as follows:

$$\min \sum_{t=1}^{T} \sum_{j:r_j<t} \left(w_j - \sum_{u=r_j}^{t} w_{j,u} \right) \tag{21}$$

$$\sum_{u=r_j}^{T} w_{j,u} = w_j \qquad j = 1, \ldots, n \tag{22}$$

$$\sum_{j=1}^{n} w_{j,u} = s_u \qquad u = 1, \ldots, T \tag{23}$$

$$\sum_{u=0}^{T} s_u^{\alpha} \leq E \tag{24}$$

$$w_{j,u} \geq 0 \qquad j = 1, \ldots, n \quad u = 1, \ldots, T \tag{25}$$

The time T is some sufficiently late time that we may be sure all jobs finish by this time in the optimal schedule. It is easy to see that T may be pseudo-polynomially bounded. The variable $w_{j,u}$ is the amount of work done on job j at time u. The objective function is just the definition of fractional flow time in a discrete setting. The constraints in line 22 express the fact that each job must be eventually finished. The constraints in line 23 define the speed s_u at time u. The constraint in line 24 is then the energy bound.

This convex program can be solved to arbitrary precision in polynomial time in the size of the program, which is pseudo-polynomial in the input size, using the Ellipsoid algorithm [14]. It is known that the response time of an optimal fractional response time schedule is within a factor of $\frac{1}{\epsilon}$ of the optimal response time schedule for a processor that is a factor of $(1+\epsilon)$ slower [6]. The proof in [6] assumes a fixed speed processor, but easily carries over to the case of a variable speed processor. Thus, for arbitrary work jobs, we can compute an $O(\frac{1}{\epsilon})$-approximate schedule given an extra factor of $(1+\epsilon)^{\alpha}$ energy.

5 Conclusions

We believe that speed scaling problems, particularly those without deadlines, is a research area where there are many combinatorially interesting problems with real application. Probably the most obvious open question that arises from this paper is whether there is a polynomial time algorithm to compute the energy optimal schedule in the case of arbitrary work jobs. More generally, it would interesting to determine whether our algorithmic approach is applicable to other speed scaling problems.

Very recently [1] considers an online variation of the flow time scheduling problem that we consider here. They consider a combined objective of the total flow time plus the energy used. They essentially show that running at a power approximately equal to the number of released but unfinished jobs is $O(1)$-competitive.

Acknowledgments: We thank Mahai Anitescu for several helpful discussions. The observation that our algorithm for equi-work jobs can be applied to arbitrary work jobs arose from discussions with Sandy Irani and John Augustine. We thank Aleksandar Ivetic for creating the java applet illustrating the change of the optimal schedule as a function of energy.

References

[1] S. Albers and H Fujiwara. Energy efficient algorithms for flow time minimization. In *Symposium on Theoretical Aspects of Computer Science*, pages 621–633, 2006.

[2] John Augustine, Chaitanya Swamy, and Sandy Irani. Optimal power-down strategies. In *Symposium on Foundations of Computer Science*, pages 530–539, 2004.

[3] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, pages 520–529, 2004.

[4] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *Symposium on Theoretical Aspects of Computer Science*, pages 460–471, 2005.

[5] Mokhtar Bazaraa, Hanif Sherali, and C. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley, 1979.

[6] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk R. Pruhs. Online weighted flow time and deadline scheduling. In *Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 36–47, 2001.

[7] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: design and modeling challenges for next generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[8] Jian-Jia Chen, Tei-Wei Kuo, and Hsueh-I Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures*, pages 338–349, 2005.

[9] Jian-Jia Chen, Tei-Wei Kuo, and Chia-Lin Yang. Profit-driven uniprocessor scheduling with energy and timing constraints. In *ACM Symposium on Applied Computing*, pages 834–840, 2004.

[10] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Symposium on Discrete Algorithms*, pages 37–46, 2003.

[11] W. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Design Automation*, pages 211–230, 2003.

[12] Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. In *International Computing and Combinatorics Conference*, pages 283–296, 2005.

[13] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer Magazine*, 34(4):52–58, 2001.

[14] I. Nesterov and Nemirovski. *Interior Point polynomial algorithms in convex programming*. Society for Industrial and Applied Mathematics, 1994.

[15] Kirk Pruhs, Rob van Stee, and Patchrawat "Patch" Uthaisombut. Speed scaling of tasks with precedence constraints. In *Proceedings of the WAOA*, pages 307–319, 2005.

[16] F. Yao, A Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[17] H.S. Yun and J. Kim. On energy-optimal voltage scheduling for fixed priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3):393–430, 2003.