Getting the Best Response for Your Erg

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

Abstract

We consider the bi-criteria problem of minimizing the average flow time (average response time) of a collection of dynamically released equi-work processes subject to the constraint that a fixed amount of energy is available. We assume that the processor has the ability to dynamically scale the speed at which it runs, as do current microprocessors from AMD, Intel, and Transmeta. We first reveal the combinatorial structure of the optimal schedule. We then use these insights to devise a relatively simple polynomial time algorithm to simultaneously compute, for each possible energy, the schedule with optimal average flow time subject to this energy constraint.

1 Introduction

Since the early 1970's the power consumption of the most common microprocessors has increased by a factor of 3 approximately every 10 years [1]. Power consumption has always been a critical issue in portable platforms such as cell phones and laptops with limited batteries (as anyone who has taken their laptop on a long flight knows). Now power consumption has become a critical issue in more traditional settings. A 1998 estimate by the Information Technology Industry Council is that computers consume 13% of the electrical power generated in the United States [1]. Further, recent estimates are that energy costs are 25% of the total cost of operating a server farm [5]. Limiting power consumption has become a first-class architectural design constraint in almost all settings [5].

Several strategies have been proposed to limit power consumption in microprocessors. Here we focus on the highest profile strategy: *dynamic scaling of speed (or voltage or power)*. Currently the dominant component of microprocessor power usage is switching loss [2]. The switching loss is roughly proportional to $V^2 f$, the voltage squared times the frequency. But V and f are not independent. There is a minimum voltage required to drive the microprocessor at the desired frequency. This minimum voltage is approximately proportional to the frequency [2]. This leads to the well known cube-root rule that speed (and equivalently, frequency) is roughly proportional to the cube-root of the power, or equivalently, that the power is proportional to the speed cubed (or frequency cubed) [2]. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. For example, a Pentium-III processor uses 9 Watts when

*Computer Science Department. University of Pittsburgh. Pittsburgh PA 15260 USA. kirk@cs.pitt.edu. Supported in part by NSF grant CCR-0098752, NSF grant ANIR-0123705, and NSF ANI-0325353.

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

‡Department of Mathematics. University of Twente. g.j.woeginger@math.utwente.nl.

run at 500 megahertz but uses 22 Watts when run at 650 megahertz [3]. Note this roughly comports with what is predicted by the cube-root rule. The speed of the microprocessor can generally be controlled by the microprocessor, the operating system, or a user application.

One natural question is then what policy should be used to set the speed, say by the operating system. The speed setting policy is symbiotically related to the scheduling policy for determining which process to run. This is a bi-criteria optimization problem. The OS wants to both optimize some Quality of Service (QoS) measure that it provides to the applications, and to minimize the energy that it uses. The way in which this problem has been formalized in the literature [7, 4] is to assume that the processes have deadlines, and then find the minimum energy schedule where all deadlines are met. However, in general computational settings, most processes do not have natural deadlines associated with them, which is why operating systems like Unix and Windows do not have deadline based schedulers. By far the most commonly used QoS measure in the computer systems literature is average flow time (average response time), which is the average over all processes, of the time that that process has to wait between when it is released until the time that is was completed.

Thus, in this paper we initiate the study of the bi-criteria problem of minimizing average flow time and minimizing energy usage. Note that these two criteria are in opposition, that is, increasing energy usage will decrease average flow time. The simplest way to formalize a bi-criteria optimization problem is to fix one parameter and to optimize the other parameter. For this problem it seems logical to us to initially fix the available energy, and then to minimize average flow time. This is certainly most logical in a setting such as a laptop where energy is provided by a battery. In this paper, we restrict our attention to the case that all processes have the same amount of work. The most important reason for our adoption of this restriction is that it lets us decouple the scheduling policy from the speed setting policy, and lets us concentrate our efforts on understanding speed scheduling. In the case of equi-work jobs, it is easy to see that the optimal scheduling policy is First-Come-First-Served. We assume that power is proportional to the speed to some power α, which generalizes the cube-root rule.

We first reveal the combinatorial structure of the optimal schedule. This structure has some surprising aspects to it. For example, intuitively the less energy available, the slower that jobs should be run in the optimal schedule. But we show that in fact this intuition is not always correct, that is, as energy decreases some jobs will 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. Fortunately, we are saved by the fact that essentially everything, including the underlying schedules, is a continuous function of energy. It also helps that over all possible amounts of available energy, that there are only linearly many possible structurally different schedules. Using these facts we obtain an $O(n^2 \log L)$ time algorithm to find all of these schedules, as well as the energy range where each schedule is optimal. Here L is range of possible energies divided by the precision that we desire. Because this problem is highly non-linear, we can get solutions where the variables are not only irrational, but don't appear to have any simple short 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. Also as a side product, these schedules can also be used to compute the schedule that optimizes energy usage subject to a constraint on average flow time.

One of our students, Aleksandar Ivetic, implemented a variation of our algorithm in Mathematica, and created a GUI in Java. The GUI lets you enter an arbitrary instance. The GUI also has a slider that lets change the available energy, and view the resulting optimal schedule. By moving the slider, you essentially get a movie of the evolving optimal schedule. The software can be found at `http://www.cs.pitt.edu/~utp/energy`. A screen capture of the GUI is shown in figure 1.

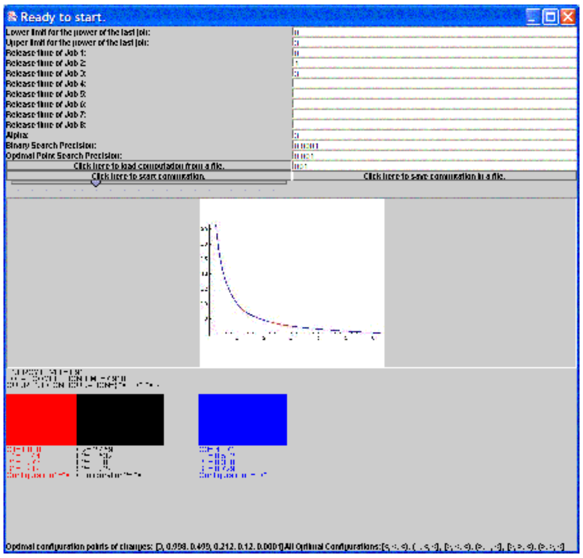

Figure 1: Screen capture of the Java GUI

We believe that this paper should be viewed as an initial investigation into a research area where there appear to be many combinatorially interesting problems with real application. We discuss this further in the conclusion.

1.1 Related Results

The way in which this problem has been formalized in the literature [7, 4] is to assume that the processes have deadlines, and then find the minimum energy schedule where all deadlines are met. In [7] a polynomial-time offline algorithm for computing the optimal schedule is presented. Further [7] gives an online algorithm with constant competitive ratio. In [4] a variation on this problem is considered. There it is assumed that the processor can be put into a lower-power sleep state, and that bringing the processor back to the on state requires a fixed amount of energy. The paper [4] gives a polynomial-time offline algorithm with an approximation ratio of 3, and an online algorithm with a constant competitive ratio.

2 Definitions, Notation, and Preliminaries

The input consists of n jobs, referenced by the integers from 1 to n. The jobs will be processed on one machine. We normalize the unit of work so that each job has unit work. Job i arrives at its release time r_i. We will assume that the jobs are labeled so that $r_1 < r_2 < ... < r_n$. Note that we assume no pair of jobs have identical release dates to ease the exposition. At the end we'll explain how the result can be easily generalized to the case that jobs can have identical release dates. A *schedule* specifies, for each time, the job that is run, and the speed at which that job is run. 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 flow time of job i, and $F = \sum_{i=1}^{n}(C_i - r_i)$ be the 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 $\alpha > 1$. 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 flow time (or equivalently, minimum average flow time) among all feasible schedules at energy level A.

It is easy to see that the optimal schedule runs jobs in First-Come-First-Served order. Thus, without loss of generality, we may only consider schedules where $C_1 < C_2 < ... < C_n$. Further, there is always an optimal schedule that runs each job at a fixed speed. This was noted in [7], and immediately follows from the convexity of the speed to power function. Therefore, to specify an optimal schedule for this problem, it suffices to specify a speed s_i for each job i. Let $x_i = 1/s_i$ be the time that job i is processed, $p_i = s_i^\alpha$ be the power for job i, and $e_i = x_i p_i = s_i^{\alpha-1}$ be the energy used by the ith job. Let $E = \sum_{i=1}^{n} e_i$ be the total energy used.

Let $\rho = s_n^\alpha$ be the power of the nth job, which we will see will play an important role in our results. To refer to a particular schedule S when the schedule in question is not clear from context, we append (S) to our notation, so $F(S)$ is the total flow time for schedule S.

If the problem is not yet clear to the reader, we refer him/her to the example instance that we use to illustrate our results in Section 3.

3 Our Results

3.1 Expressing the Problem as a Convex Program

We note that the problem of minimizing total flow time subject to an energy constraint can be expressed as the following convex program CP:

$$\begin{aligned}
&\text{minimize} && C = \sum_{i=1}^{n} C_i \\
&\text{subject to} && \\
&\sum_{i=1}^{n} \frac{1}{x_i^{\alpha-1}} \le A && \\
&x_i = C_i - \max\{r_i, C_{i-1}\} && \text{for } i = 1, ..., n \\
&C_i > r_i && \text{for } i = 1, ..., n \\
&C_i < C_{i+1} && \text{for } i = 1, ..., n
\end{aligned}$$

We assume that there are dummy variables $C_0 = 0$ and $C_{n+1} = \infty$. While CP is not strictly speaking convex, it can be made convex by introducing new variables t_i, and the constraints $t_i \ge r_i$, $t_i \ge C_{i-1}$, $x_i = C_i - t_i$, and $x_i \ge 1/A$. Thus this problem can be solved in polynomial time by the Ellipsoid algorithm since the gradients of the constraint and objective functions are efficiently computable [6]. However, this would only solve the problem for one energy bound, would be quite complex, and is not guaranteed to run in lower order polynomial time.

3.2 Configuration Curves

The program CP prompts us to define what we call configurations. A *configuration* ϕ maps each job i, $1 \leq i \leq n-1$, to a relation in $\{<, =, >\}$. 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 flow time for this schedule), among those schedules in configuration ϕ that use less than 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 flow time.

As an example, consider the 3 job instance with release times 0, 1 and 3, and $\alpha = 3$. There are five configuration curves that are part of $M(A)$. The configuration curves for the configurations $>><$, $><<$ and $<<<$ are shown in Figure 2 from left to right in this order. Note that configuration curves are in general only defined over a subrange of possible energies. The superposition of all five configuration curves that make up $M(A)$ are show in Figure 3. We recommend viewing these figures in color. The corresponding configurations/schedules are shown in Figure 4. Note that in Figure 4 that the job released at time 1 is run at faster speeds as energy decreases from 1.282 to 1.147.

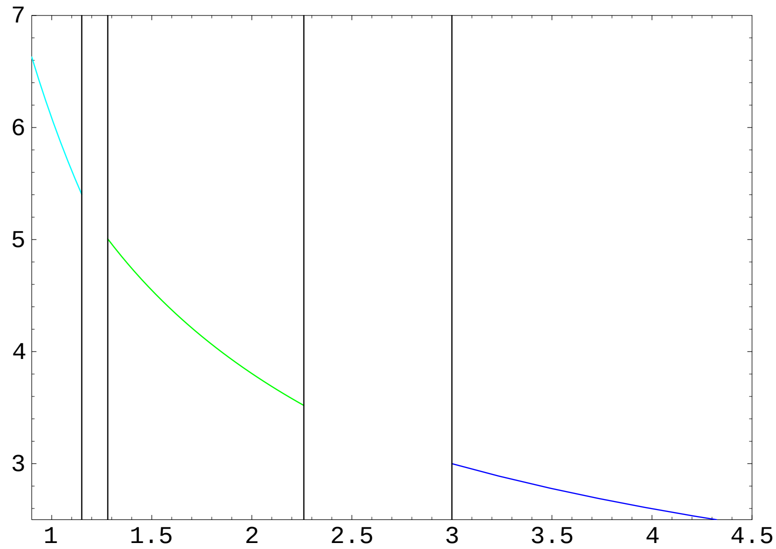

Figure 2: The configuration curves for the configurations $>><$, $><<$ and $<<<$

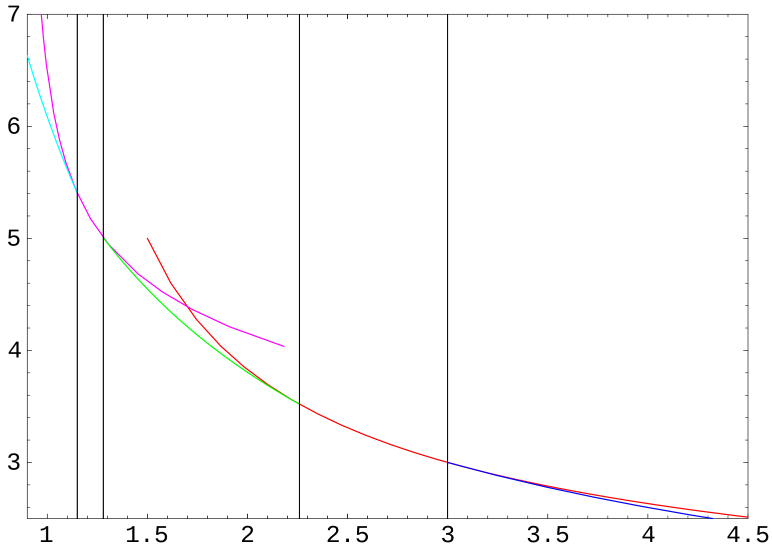

Figure 3: The superposition of the five configuration curves that make up $M(A)$.

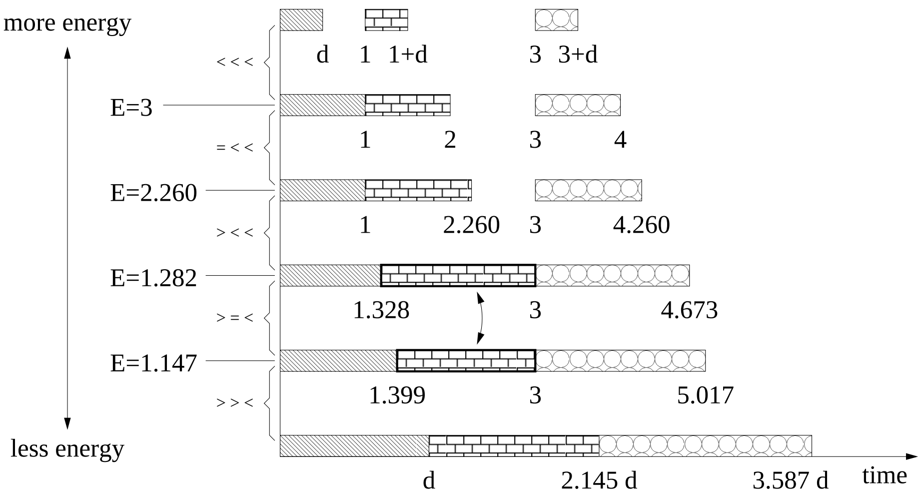

Figure 4: Optimal schedules at different amounts of available energy

3.3 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 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 speed. As we decrease A we need to explain:

- how to compute $M_\phi(A)$ for any given configuration ϕ,
- how to recognize when the optimal configuration changes from a configuration ϕ to a configuration ϕ', and
- how to find the new optimal schedule when we switch configurations.

If we can accomplish there goals then we could trace out $M(A)$ by continuously decreasing A, or equivalently ϕ.

Actually, rather than working with the energy bound A, it will prove more convenient to work with the power ρ of job n. We will eventually show in Lemma 8 that moving continuously from a high ρ to a low ρ is equivalent to continuously moving from a high A to a low A. Eventually we will explain how to make this algorithm efficient by discretely changing A and ρ.

3.4 How to Find the New Optimal Schedule When We Switch Configurations

We tackle this last goal first by showing that the optimal schedule is unique for each energy. Thus when two configuration curves intersect on the lower envelope for some energy A, we know that the underlying schedules are identical. Thus the optimal schedule for the new configuration is the same as the optimal schedule for the old configuration.

Lemma 1. *The optimal schedule is unique given that $\alpha > 1$.*

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 flow 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 righthand 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 $(e_i(S) + e_i(T))/2 = (\frac{1}{x_i(S)^{\alpha-1}} + \frac{1}{x_i(T)^{\alpha-1}})/2$. □

3.5 How to Compute $M_\phi(A)$

We now consider our second goal. That is, 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 ρ, and a fixed configuration ϕ, finding the optimal schedule among those schedules with configuration ϕ and power ρ on job n. We define a *group* as a maximal substring of jobs where $\phi(i)$ is $>$. More precisely $a, a+1, \dots, b$ 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 $>$.

This group is *open* if $\phi(b)$ is $<$, and this group is *closed* if $\phi(b)$ is $=$.

Lemmas 2, 3, and 4 establish a relationship between speed s_i and speed s_{i+1} depending on whether $C_i > r_{i+1}$, $C_i < r_{i+1}$, or $C_i = r_{i+1}$.

Lemma 2. *If i is a job in the optimal schedule for energy A such that $C_i > r_{i+1}$, then $s_i^\alpha = \rho + s_{i+1}^\alpha$.*

Proof. Let ϵ be a small number such that $C_i + \epsilon > r_{i+1}$. Note that ϵ is allowed to be either positive or negative. Consider the result of increasing x_i by ϵ, decreasing x_{i+1} by ϵ, and decreasing x_n by ϵ. This does not change the total completion time as C_i increases by ϵ, C_n decreases by ϵ, and C_{i+1} and all other C_j's remain unchanged. The change in the energy used, $\Delta E(\epsilon)$, is

$$(\frac{1}{x_i+\epsilon})^{\alpha-1} - (\frac{1}{x_i})^{\alpha-1} + (\frac{1}{x_{i+1}-\epsilon})^{\alpha-1} - (\frac{1}{x_{i+1}})^{\alpha-1} + (\frac{1}{x_n-\epsilon})^{\alpha-1} - (\frac{1}{x_n})^{\alpha-1} \tag{1}$$

Since the optimal schedule is unique (Lemma 1), $\Delta E(\epsilon)$ must be positive. Otherwise, we could reinvest the energy saved by this change to obtain a schedule with a better total completion time. Hence the derivative $\Delta E'(\epsilon)$ evaluated at $\epsilon = 0$ must be 0.

$$\Delta E'(\epsilon) = \frac{-(\alpha-1)}{(x_i+\epsilon)^\alpha} + \frac{\alpha-1}{(x_{i+1}-\epsilon)^\alpha} + \frac{\alpha-1}{(x_n-\epsilon)^\alpha} \tag{2}$$

Substituting $\epsilon = 0$ and solving for $\Delta E'(0) = 0$ we get that $\frac{1}{x_i^\alpha} = \frac{1}{x_n^\alpha} + \frac{1}{x_{i+1}^\alpha}$ or equivalently, $s_i^\alpha = s_n^\alpha + s_{i+1}^\alpha$. □

Lemma 3. *If i is a job in the optimal schedule for energy A such that $C_i < r_{i+1}$, then $s_i^\alpha = \rho$.*

Proof. Let ϵ be a small number such that $C_i + \epsilon < r_{i+1}$. Note that ϵ is allowed to be either positive or negative. Consider the result of decreasing x_i by ϵ, and increasing x_n by ϵ. This does not change the total completion time as C_i decreases by ϵ and C_n increases by ϵ, and all other C_j's remain unchanged. The change in the energy used, $\Delta E(\epsilon)$, is

$$(\frac{1}{x_i-\epsilon})^{\alpha-1} - (\frac{1}{x_i})^{\alpha-1} + (\frac{1}{x_n+\epsilon})^{\alpha-1} - (\frac{1}{x_n})^{\alpha-1} \tag{3}$$

Since the optimal schedule is unique (Lemma 1), $\Delta E(\epsilon)$ must be positive. Otherwise, we could reinvest the energy saved by this change to obtain a schedule with a better total completion time. Hence the derivative $\Delta E'(\epsilon)$ evaluated at $\epsilon = 0$ must be 0.

$$\Delta E'(\epsilon) = \frac{\alpha - 1}{(x_i - \epsilon)^\alpha} + \frac{-(\alpha - 1)}{(x_n + \epsilon)^\alpha} \tag{4}$$

Substituting $\epsilon = 0$ and solving for $\Delta E'(0) = 0$ we get that $\frac{1}{x_i^\alpha} = \frac{1}{x_n^\alpha}$ or equivalently, $s_i^\alpha = s_n^\alpha$. □

Lemma 4. *If i is a job in the optimal schedule for energy A such that $C_i = r_{i+1}$, then $\rho \leq s_i^\alpha \leq \rho + s_{i+1}^\alpha$.*

Proof. First we show that $s_i^\alpha \leq s_n^\alpha + s_{i+1}^\alpha$ Assume to reach a contradiction that there is an optimal solution with $-s_i^\alpha + s_{i+1}^\alpha + s_n^\alpha < 0$. Consider the transformation that increases x_i by a small $\epsilon > 0$, and that decreases x_{i+1} by ϵ, and decreases x_n by ϵ. This does not change the total completion time. We now argue that this decreases the energy used. This is sufficient because one could then get a contradiction by reinvesting this saved energy into the last job to improve the total completion time. Note that the transformation will bring us into a new configuration because $C_i + \epsilon > C_i = r_{i+1}$. The change in energy used, $\Delta E(\epsilon)$, is then given by Equation (1), and its derivative, $\Delta E'(\epsilon)$, is given by Equation (2). Substituting $\epsilon = 0$ we get that

$$\Delta E'(0) = -\frac{\alpha - 1}{x_b^\alpha} + \frac{\alpha - 1}{x_{b+1}^\alpha} + \frac{\alpha - 1}{x_n^\alpha} = (\alpha - 1)(-s_b^\alpha + s_{b+1}^\alpha + s_n^\alpha) < 0$$

where the inequality follows from the assumption above. Thus, we have a contradiction as claimed.

Next we show that $s_n^\alpha \leq s_i^\alpha$. Assume to reach a contradiction that there is an optimal solution with $s_i^\alpha - s_n^\alpha < 0$. Consider the transformation that decreases x_i by a small $\epsilon > 0$, increases x_n by ϵ. This does not change the total completion time. We now argue that this decreases the energy used. This is sufficient because one could then get a contradiction by reinvesting this saved energy into the last job to improve the total completion time. Note that the transformation will bring us into a new configuration because $C_i - \epsilon < C_i = r_{i+1}$. The change in energy used, $\Delta E(\epsilon)$, is then given by Equation (3), and its derivative, $\Delta E'(\epsilon)$, is given by Equation (4). Substituting $\epsilon = 0$ we get that

$$\Delta E'(0) = \frac{\alpha - 1}{x_i^\alpha} + \frac{-(\alpha - 1)}{x_n^\alpha} = (\alpha - 1)(s_i^\alpha - s_n^\alpha) < 0$$

where the inequality follows from the assumption above. Thus, we have a contradiction as claimed. □

Lemma 5 states how to compute the speeds of the jobs in a group given the speed of the last job of the group.

Lemma 5. *If $a, a+1, \ldots, b$ are jobs in a group in the optimal schedule for energy level A, then $s_i^\alpha = s_b^\alpha + (b - i)\rho$ for $i = a, \ldots, b$.*

Proof. This is an immediate application of Lemma 2. □

It should now be clear how to compute the speeds/powers of the jobs $a, a+1, \ldots, b$ in an open group. The power of the last job in the group is ρ, and the powers of the earlier jobs in the group are given by Lemma 5. To compute the speeds of the jobs in a closed group is a bit more involved. Lemma 6 establishes a relationship between the speed of the speeds of the jobs in a closed group and the length of the time period when the jobs in this group are run.

Lemma 6. *If $a, a+1, \ldots, b$ are jobs in a closed group in the optimal schedule for energy level A, then $\sum_{i=a}^{b} \frac{1}{(s_b^\alpha+(b-i)\rho)^{1/\alpha}} = r_{b+1} - r_a$.*

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,

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

where the last equality follows from Lemma 5, □

Lemma 6 gives an implicit definition of s_b as a function of ρ and hence of the other s_i's in this closed group (using Lemma 5). However, it is hard, if at all possible, to determine the closed form for s_b. Lemma 7 tells us that $\sum_{i=a}^{b} \frac{1}{(s_b^\alpha+(b-i)\rho)^{1/\alpha}}$ is strictly increasing as a function of s_b. Therefore, one can determine s_b from ρ by binary search on s_b. We can then compute the speed of other jobs in this closed group using Lemma 5.

Lemma 7. *When ρ is fixed, $\sum_{i=a}^{b} \frac{1}{(s_b^\alpha+(b-i)\rho)^{1/\alpha}}$ decreases as s_b increases.*

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

Finally, we show that continuously decreasing ρ 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 8. *In the optimal schedules, then energy bound A is a strictly increasing continuous function of ρ, and similarly, ρ is a strictly increasing continuous function of A.*

Proof. We first prove that there is a bijection between ρ and A in the optimal schedules, as well as in the optimal schedules restricted to a particular configuration. The fact that a fixed ρ is mapped into a unique energy should be obvious from our development to date. That two different ρ's can not map to optimal schedules with the same energy follows from Lemma 1.

Since the function from ρ to A is obviously continuous, it then follows that the function from ρ 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 $\leq$, and ρ is large. If A is very small, then the optimal configuration is all $\geq$, and ρ is small. □

3.6 How to Recognize When the Optimal Configuration Changes

We now tackle the second of our three goals: how to recognize when the optimal configuration changes from a configuration ϕ to a configuration ϕ'. The formulas for computing the speeds of the jobs in the previous section may not yield a configuration equal to ϕ. In particular, this could happen if one of the $<$ constraints is violated, or if there is a last job b in some closed group with $s_b^\alpha > s_{b+1}^\alpha + \rho$. In a non-degenerate case, the configuration obtained from ρ will be different from ϕ by exactly one constraint $\phi(i)$. The next configuration ϕ' is obtained from ϕ by changing $\phi(i)$. If $\phi(i)$ is $<$, it should be changed to $=$. If $\phi(i)$ is $=$, it should be changed to $>$. In a degenerate case, all violating $\phi(i)$'s need to be changed.

3.7 Implementation Details and Time Complexity

To construct an efficient implementation of this algorithm we need to change ρ in a discrete fashion. This can be accomplished using binary search to find the next value for ρ when the optimal configuration changes. The condition for determining whether the current configuration is optimal for a particular ρ, described in the last subsection, can be computed in linear time. Thus in something like time $O(n \log L)$ we can find the next configuration curve on the lower envelope, where L is something like the range of possible values of ρ divided by our desired accuracy. It is easy to see that 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 something like $O(n^2 \log L)$.

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 Lemmas 3 and 5.

4 Conclusions and Future Directions

We believe that paper suggest several interesting algorithmic and combinatorial research directions. The two most obvious directions are to consider offline algorithms for arbitrary length jobs, and to consider online algorithms.

First let us consider offline algorithms for arbitrary length jobs. A simple exchange argument shows that all optimal schedules maintain the invariant that they run a job with minimal remaining work. Note that this does not uniquely define the job ordering as the remaining work depends on the speed at which the jobs were run in the past. While many of our statements carry to the case of arbitrary job lengths, many do not. In particular,

- The most obvious mathematical programs corresponding to CP are no longer convex.
- When a job is broken into multiple pieces due to preemptions, our notion of configurations breaks down. See Figure 5.
- It is not clear that the lower envelope is of polynomial size.
- There will be transitions on the lower envelope corresponding to reordering of jobs and preemptions. Further optimal schedules can change in a non-continuous fashion at these transitions. See Figure 5.

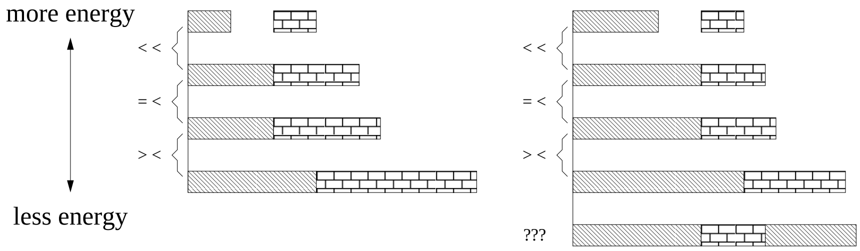

Figure 5: Optimal configurations at different amounts of available energy when jobs have unit work (left) and arbitrary amount of work (right)

Of course any algorithm implemented in an operating system must be online. Although an analysis of online algorithms seems at least a little tricky for several reasons. Firstly, it is not quite clear how to most reasonably formalize the problem. Secondly, the fact that energy is very non-local makes dealing with equal energy schedules a bit messy. Hopefully our revelation of the structure of the optimal schedule in this paper will prove useful in this regard.

Acknowledgments: we thank Mahai Anitescu for pointing out that the mathematical program CP could be made convex, and for several helpful discussions. We thank Aleksandar Ivetic for implementing our algorithm.

References

[1] K. Azar. Managing power requirements in the electronics industry. *Electronics Cooling Magazine*, 6(4), 2000.

[2] 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), 2000.

[3] D. Grunwald, P. Lewis, and K. Farkas. Policies for dynamic clock scheduling. In *USENIX Symposium on Operating Systems Design and Implementation*, 2000.

[4] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Symposium on Discrete Algorithms*, 2003.

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

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

[7] 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.