Dynamic Speed Scaling to Manage Energy and Temperature

Nikhil Bansal
IBM T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY, 10598, USA
nikhil@us.ibm.com

Tracy Kimbrel
IBM T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY, 10598, USA
kimbrel@us.ibm.com

Kirk Pruhs*
Dept. of Computer Science
University of Pittsburgh
Pittsburgh , PA 15260
kirk@cs.pitt.edu

Abstract

We first consider online speed scaling algorithms to minimize the energy used subject to the constraint that every job finishes by its deadline. We assume that the power required to run at speed s is $P(s) = s^\alpha$. We provide a tight α^α bound on the competitive ratio of the previously proposed Optimal Available algorithm. This improves the best known competitive ratio by a factor of 2^α. We then introduce a new online algorithm, and show that this algorithm's competitive ratio is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$. This competitive ratio is significantly better and is approximately $2e^{\alpha+1}$ for large α. Our result is essentially tight for large α. In particular, as α approaches infinity, we show that any algorithm must have competitive ratio e^α (up to lower order terms).

We then turn to the problem of dynamic speed scaling to minimize the maximum temperature that the device ever reaches, again subject to the constraint that all jobs finish by their deadlines. We assume that the device cools according to Fourier's law. We show how to solve this problem in polynomial time, within any error bound, using the Ellipsoid algorithm.

1. Introduction

The power requirements of computing devices have been increasing exponentially. Since the early 1970s, power densities in microprocessors have doubled every three years [13]. Limiting power requirements is a critical issue for two reasons:

- **Energy**: The energy used by a computing device is the integral over time of power. This is particularly a problem in mobile devices that rely on batteries for energy, and will become even more of a problem in the future since battery capacities are increasing at a much slower rate than power requirements.
- **Temperature**: The energy used in computing devices is in large part converted into heat. Exponentially rising cooling costs threaten the computer industry's ability to deploy new systems. In fact, in May Intel abruptly announced that it had scrapped the development of two new computer chips (code named Tejas and Jayhawk) for desktops/servers in order to market a more efficient chip technology more than a year ahead of the original schedule. Analysts said the move showed how eager the world's largest chip maker was to cut back on the heat its chips generate. Intel's method of increasing chip speed was beginning to require expensive and noisy cooling systems for computers [16]. Current estimates are that cooling solutions are rising at \$1 to \$3 per watt of heat dissipated [13].

Power is now recognized as a first-class design constraint for modern computing devices [9]. There is an extensive literature on power management in computing devices. Overviews can be found in [2, 9, 15].

Both in academic research and practice, dynamic voltage/frequency/speed scaling is the dominant technique for power management. Speed scaling involves dynamically

* Supported in part by NSF grants CCR-0098752, ANI-0123705, and CNS-0325353.

changing the speed of the processor. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. However, there is an inherent conflict between power reduction and performance; in general, the more power that is available, the better the performance that can be achieved. This gives rise to dual objective optimization problems since one seeks to optimize both performance, and either energy or temperature.

1.1. Background

Power in CMOS based devices, which will likely remain the dominant technology for the near term future, has three components: switching loss, leakage loss, and short circuit loss [2, 9]. Switching loss measures the power consumption from charging and discharging gates. The switching loss is roughly proportional to sV^2, where s is the speed (clock frequency), and V is the voltage. But s and V 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 the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$, i.e., the power is proportional to the cube of the speed [2]. Currently, switching loss makes up the majority of the power used by computing devices [9].

Most of the research in the literature to date generalizes the cube root rule so that the power $P(s) = s^\alpha$ for some constant $\alpha > 1$, or even more generally, to the case that $P(s)$ is strictly convex. The assumption that $P(s)$ is strictly convex means that the more slowly a task is run, the less energy is used to complete that task.

Some modern processors are able to sense their own temperatures, and thus can be slowed down or shut down so the processor temperature will stay below its thermal threshold [13].

Fourier's Law law of heat conduction states that the rate of cooling is proportional to the difference in temperature between the object and the environment. We assume that the environment has a fixed temperature, and that temperature is scaled so that the environmental temperature is zero. A first order approximation for rate of change $T'(t)$ of the temperature $T(t)$ over time t is then $T'(t) = aP(t) - bT(t)$, where $P(t)$ is the supplied power at time t, and a and b are constants [12]. Equivalently, the power is given by $P(t) = (T'(t) + bT(t))/a$. Thus, if the cube root rule applies, then the instantaneous speed of the processor is given by $(T'(t) + bT(t)/a)^{1/3}$.

1.2. Prior Research

Theoretical worst-case investigation of speed scaling algorithms was initiated in [17]. The problem is to schedule a given collection of tasks, where each task i has a release time r_i when it arrives into the system, an amount of work w_i that must be performed to complete the task, and a deadline d_i for completing this work. In the online version of the problem, the scheduler learns about a task only at its release time; at this time, the scheduler also learns the exact work requirement and the deadline of the job. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks [3]. A schedule specifies which task to run at each time, and at what speed that task should be run. The schedule must be feasible with respect to the deadlines; that is, each task must finish by its deadline. This is always possible since the processor can run at any speed. Preemption is allowed, that is, the scheduler may suspend a task and then later restart the task from the point of suspension.

The problem considered in [17] was to minimize the total energy used to complete all tasks subject to the deadline feasibility constraints. [17] gives an optimal offline greedy polynomial-time algorithm for this problem for any power exponent $\alpha \geq 1$. We call this algorithm YDS, which we now describe. Let $w(t_1, t_2)$ denote the work that has release time $\geq t_1$ and has deadline $\leq t_2$. The *intensity* of the interval $[t_1, t_2]$ is then $w(t_1, t_2)/(t_2 - t_1)$. The YDS algorithm repeats the following steps: Let $[t_1, t_2]$ be the maximum intensity interval. The processor will run at speed $w(t_1, t_2)/(t_2 - t_1)$ during $[t_1, t_2]$. Then the instance is modified as if the times $[t_1, t_2]$ didn't exist. That is, all deadlines $d_i > t_1$ are reduced to $\max(t_1, d_i - (t_2 - t_1))$, and all release times $r_i > t_1$ are reduced to $\max(t_1, r_i - (t_2 - t_1))$, and the process is repeated. Given the speed as a function of time as determined by this procedure, YDS always runs the released, unfinished job with the earliest deadline. The YDS algorithm also minimizes the maximum speed, over all times in the schedule, that the processor runs at that time, since this corresponds to the limit as α increases to infinity.

[17] proposed two simple online algorithms. The online algorithm Average Rate (AVR) runs each task i at a rate of $w_i/(d_i - r_i)$. The online algorithm Optimal Available (OA) at any point of time schedules the unfinished work optimally (say using YDS) under the assumption that no more tasks will arrive. [17] state a lower bound of α^α on the competitive ratio for AVR and OA. [17] prove that the competitive ratio of AVR is at most $2^\alpha \alpha^\alpha$. The analysis of the AVR algorithm in [17] is a rather complicated spectral analysis, which does not easily yield an intuitive explanation of AVR's performance. [17] does not provide any upper bound analysis of OA. [17] state a lower bound of $10/9$ on the competitiveness of any online algorithm (presumably for the case $\alpha = 2$). This lower bound instance consists of two jobs.

[8] studies online speed scaling algorithms to minimize

energy usage in a setting where the device also has a sleep state, and gives an offline polynomial-time 2-approximation algorithm. [8] also gives an online algorithm A that uses as a subroutine an algorithm B for pure dynamic speed scaling. If B is additive and monotone (as AVR and OA are) then A is $\max\{\alpha(c+1)+2, 4\}$ competitive, where c is the competitive ratio of B. Thus in case that the cube-root rule holds, using the analysis of AVR from [17], one obtains an algorithm with competitive ratio of at most 653. These results have been extended to the case of multiple slow-down states in [1].

[11] give an efficient algorithm for the problem of minimizing the average flow time of a collection of dynamically released equi-work processes subject to the constraint that a fixed amount of energy is available.

1.3. Our Contribution

We first extend the results in [17] for online algorithms for energy minimization. We explicitly give instances that that show that the competitive ratios of AVR and OA are at least α^α. We then provide a tight α^α bound on the competitive ratio of OA. This improves the bound on the best known competitive ratio by a factor of 2^α. Our analysis of OA is relatively straight-forward potential function argument. In the case that the cube root rule applies, this lowers the best known competitive ratio from 216 to only 27. For the problem considered in [8], where there is a sleep state, if the cube root rule applies this lowers the best known competitive ratio from 653 to a mere 86. We show a general lower bound on the competitive ratio for any randomized online algorithm of $\Omega((4/3)^\alpha)$. Again this instance contains two jobs. As more jobs are added, the combinatorial analysis quickly becomes complicated. Thus it seems one is unlikely to find a strongly competitive algorithm. Instead, it seems that the right question that this lower bound raises is whether there is an online algorithm with competitive ratio $O(1)^\alpha$. We answer this question in the affirmative, by introducing a new online algorithm, and showing that this algorithm's competitive ratio is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$. Note that for $\alpha \geq 2$, this competitive ratio is at most $8e^\alpha$. Further, the competitive ratio of this algorithm is better than the the competitive ratios of OA and AVR for $\alpha \geq 5$. We also that the exponent e cannot be improved. In particular, if we consider the case when α approaches ∞, the power consumption is completely determined by the maximum speed at which the processor ever runs. We show a general lower bound of e on the competitive ratio of any deterministic algorithm for minimizing the maximum speed (hence a lower bound of essentially e^α on the power consumed). Finally, we also show that our algorithm e competitive for the problem of minimizing the maximum speed at which the processor ever runs.

We then turn to the problem of dynamic speed scaling to minimize the maximum temperature that the device ever reaches, again subject to the constraint that all jobs finish by their deadlines. We assume that the device cools according to Fourier's law. We show that this problem can be posed as a convex optimization problem. Convex optimization problems can be solved arbitrarily well in polynomial time if one can compute a separating hyperplane for a violated constraint in polynomial time. To accomplish this for our temperature problem, we must solve the key subproblem of determining the maximum work that can be accomplished during a fixed period with a fixed starting and ending temperature. We show how to use techniques from calculus of variations to solve this subproblem. As a disclaimer, we totally ignore issues of numerical stability, which would certainly be necessary to address if one were to implement this algorithm on a finite precision machine.

2. Online Algorithms for Minimizing Energy

2.1. A Tight Analysis of Optimal Available

We now determine the competitive ratio of the Optimal Available algorithm proposed in [17]. Recall that Optimal Available (OA) simply computes the YDS schedule based on the current knowledge of jobs. We start by giving an explicit lower bound instance for OA and AVR.

Lemma 1 *The competitive ratio of AVR and OA is at least* α^α.

Proof: The instance is defined as follows: All jobs have the same deadline n. For $i = 0, 1, \ldots, n-1$, a job of size $(1/(n-i))^{1/\alpha}$ arrives at time i. The offline algorithm works on the job that arrives at time i during the interval $(i, i+1]$. It is easy to check that the offline cost is $\sum_{i=1}^{n} 1/i$.

Let $w(x)$ denote the backlogged workload at time x under OA. By definition, OA works at rate $w(x)/(n-x)$ during time $(x, x+1]$. Thus $w(x+1) - w(x) = 1/(n-x)^{1/\alpha} - w(x)/(n-x)$. The solution to this recurrence is easily seen to be $\alpha(n-x)^{1-1/\alpha}$ (in the continuous case). Thus the total online cost is $\sum_{i=0}^{n-1} \alpha^\alpha 1/(n-i)$, which is exactly α^α times the offline cost. ■

Before we prove the upper bound, we need the following lemma.

Lemma 2 *Let* $q, r, \delta \geq 0$ *and* $\alpha \geq 1$. *Then* $(q+\delta)^{\alpha-1}(q - \alpha r - (\alpha-1)\delta) - q^{\alpha-1}(q-\alpha r) \leq 0$.

Proof: We need to show that

$$(q+\delta)^{\alpha-1}(q-\alpha r) - (q+\delta)^{\alpha-1}(\alpha-1)\delta - q^{\alpha-1}(q-\alpha r) \leq 0$$

or

$$(q-\alpha r)[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \leq 0.$$

Since $[(q+\delta)^{\alpha-1} - q^{\alpha-1}] \geq 0$, it suffices to show that

$$q[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \leq 0.$$

Substituting $\delta = zq$, the left hand side of the above reduces to

$$q^{\alpha}[(1+z)^{\alpha-1} - 1] - q^{\alpha}[(1+z)^{\alpha-1}(\alpha-1)z]$$

Thus we just need to show that for $z \geq 0$,

$$(1+z)^{\alpha-1} - 1 - (1+z)^{\alpha-1}(\alpha-1)z \leq 0.$$

Differentiating this with respect to z, we get $((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z] + (1+z)^{\alpha-1}(-\alpha+1)) = ((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z-(1+z)] = -\alpha(\alpha-1)z(1+z)^{\alpha-2} \leq 0$. Thus, the maximum of this express is at $z = 0$, where the expression evaluates to 0. ∎

Theorem 3 *The algorithm Optimum Available (OA) is α^{α}-competitive.*

Proof: Let s_{on} (resp. s_{opt}) denote the speed at which OA (resp. the optimum algorithm) works at time t. We will define a potential function $\phi(t)$ that satisfies the following two properties:

- Let $\Delta(\phi(t))$ denote the change in the potential due to the arrival of a job at time t. Then $\Delta(\phi(t)) \leq 0$.
- Between arrivals, $s_{on}^{\alpha}(t) - \alpha^{\alpha} s_{opt}^{\alpha}(t) + \frac{d}{dt}\phi(t) \leq 0$.

It is easily seen that these two properties imply the α^{α} competitiveness of OA.

Before we can define the potential function we need to introduce some notation. For notational convenience, we always let the current time be 0. A time t will refer to t time units in the future from the current time. Let $s(t)$ denote the rate at which the online algorithm would be working after t time units have passed, if no new jobs ever come after the current time. Since OA simply computes the YDS schedule based on the current knowledge of jobs, the rates $s(t)$ are computed as follows. Let $w(t, t')$ denote the work under the online algorithm that is available right now and has deadline $> t$ and $\leq t'$ time units from now. We will refer to $w(t, t')/(t' - t)$ as the *density* of interval $[t, t']$. Consider a sequence of times defined inductively as follows. Let $t_0 = 0$. Let t_i denote the smallest time such that $w(t_{i-1}, t_i)/(t_i - t_{i-1}) = \max_{t' > t_{i-1}} w(t_{i-1}, t')/(t' - t_{i-1})$. Thus t_1 is the smallest time when the density of jobs between 0 and t_1 is maximized and so on. It is easy to see that $s(t) = s(t_i)$ for $t_i < t \leq t_{i+1}$ for all $i \geq 0$. We will call the interval $(t_i, t_{i+1}]$, a *critical* interval and denote it by I_i. Finally, let $w'(t, t')$ denote the work under offline at the current time that has deadline $> t$ and $\leq t'$. For notational convenience, let s_i denote $s(t_i)$. Let $w_{i,j}$ denote $w(t_i, t_j)$ and similarly let $w'_{i,j}$ denote $w'(t_i, t_j)$.

We define the potential function

$$\phi = \alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{i,i+1} - \alpha w'_{i,i+1})$$

Note that by definition the online algorithm is always working at the rate s_0 (i.e. $s_{on} = s_0$). If the current critical interval finishes, then the algorithm enters the next critical interval. The indices of the critical deadlines shift by one and the new speed s_0 is that which was previously s_1. Also note that the potential is continuous as a critical interval finishes and we move to the next one. As the current critical interval finishes i.e. t_1 approaches 0, both $w_{0,1}$ and $w'_{0,1}$ approach 0, as both algorithms have to finish this work by time t_1. So the contribution of the first term approaches 0 as the current interval is about to finish.

It is easy to see that s_i is a non-increasing sequence, as follows. If $s_{i+1} > s_i$, then this contradicts that s_i was the critical density at time t_i, since in this case $w(t_i, t_{i+2})/(t_{i+2} - t_i) > w(t_i, t_{i+1})/(t_{i+1} - t_i) = s_i$.

Working case: Let us first consider when the algorithm is simply working on some job in the first interval. Then we know that only $w_{0,1}$ is decreasing at the rate s_0. In particular, each s_i is fixed (including s_0), and $w_{i,i+1}$ is fixed for $i \geq 1$. Let k be the smallest number ≥ 0 such that $w'_{k,k+1} \neq 0$. Assume without loss of generality that the offline schedule always executes the job with the earliest deadline. Then we have that $w'_{k,k+1}$ decreases at rate s_{opt}.

Thus we have to show the non-positivity of

$$s_0^{\alpha} - \alpha^{\alpha} s_{opt}^{\alpha} + \frac{d}{dt}\alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{i,i+1} - \alpha w'_{i,i+1})$$

This evaluates to $s_0^{\alpha} - \alpha^{\alpha} s_{opt}^{\alpha} + (-\alpha s_0^{\alpha-1} s_0 + \alpha^2 s_k^{\alpha-1} s_{opt})$. Since $s_k \leq s_0$, this is at most $(1-\alpha)s_0^{\alpha} + \alpha^2 s_0^{\alpha-1} s_{opt} - \alpha^{\alpha} s_{opt}^{\alpha}$. Let $z = s_0/s_{opt}$, then showing the non-positivity of the quantity above is equivalent to showing that the polynomial $(1-\alpha)z^{\alpha} + \alpha^2 z^{\alpha-1} - \alpha^{\alpha}$ is never positive for $z \geq 0$. At $z = 0$, it evaluates to $-\alpha^{\alpha}$ which is < 0. At $z = \infty$, it is $-\infty$. If we set the derivative to 0, we find $z = \alpha$, where the polynomial has the value 0. This completes the analysis for the intervals in which work is executed between arrivals.

Arrival Case: Consider the arrival of a job of size x with deadline t time units in the future. Let i be such that $t_i < t \leq t_{i+1}$. We will show that the change in potential caused by this arrival is non-negative. Observe that each s_j for $j \geq i+1$ is unchanged by the way these rates are defined.

First we consider the simplest case when the critical intervals are unchanged (i.e., only the values of the critical densities change). Hence, the only effect the new job has is to increase the density of the interval $I_i = (t_i, t_i + 1)$ to $(w(t_i, t_{i+1}) + x)/(t_{i+1} - t_i)$ The change in the potential function in this case is

$$
\begin{aligned}
&\alpha\left(\frac{w(t_i,t_{i+1})+x}{t_{i+1}-t_i}\right)^{\alpha-1}(w(t_i,t_{i+1})-\alpha w'(t_i,t_{i+1})\\
&-\alpha\left(\frac{w(t_i,t_{i+1})}{t_{i+1}-t_i}\right)^{\alpha-1}(w(t_i,t_{i+1})-\alpha w'(t_i,t_{i+1}))\\
&-\alpha(\alpha-1)\left(\frac{w(t_i,t_{i+1})+x}{t_{i+1}-t_i}\right)^{\alpha-1}x \qquad (1)
\end{aligned}
$$

The non-positivity of the expression above follows by setting $q = w(t_i, t_{i+1})$, $\delta = x$ and $r = w'(t_i, t_{i+1})$ in Lemma 2.

We now consider the more interesting case when the arrival of a job might change the critical intervals. While this new job may radically change the structure of the critical intervals, we show that we can think of this change as a sequence of smaller changes, where each smaller change only affects two critical intervals. To see this, imagine the size of the new job increasing starting from 0. For some size $x' \le x$, one of the following two events must occur:

- Either the interval I_i remains a critical interval and its density becomes equal to that of I_{i-1}. In particular, x' is such that $s_{i-1} = (w(t_i, t_{i+1}) + x')/(t_{i+1} - t_i)$.
- Else, the interval I_i splits into two critical intervals $I'_i = (t_i, t']$ and $I''_i = (t', t_{i+1}]$ for some $t \le t' < t_{i+1}$. Since x' is the smallest such size, the densities of I'_i and I''_i are identical and equal to $(w(t_i, t_{i+1}) + x')/(t_{i+1} - t_i)$.

Now we can imagine the original job of size x to consist of two jobs, such that both arrive at the same time and have the same deadline t, but one has size x' and the other has size $x - x'$. We can thus first analyze the change in potential for x' and then repeat the procedure for $x - x'$. Thus, we only need to consider the change in potential for a job of size x' that causes one of the two events described above. Finally, observe that $x' > 0$ (and the argument above is not void), since t_{i+1} is defined as the *earliest* time after t_i for which $w(t_i, t_{i+1})/(t_{i+1} - t_i) = s_i$. Hence, a non-zero new work is needed to split the interval in to two pieces.

In the first case, the only change in the potential function is due to the change of density of I_i. This change is identical to Equation 1 with x replaced by x'.

In the second case, again the change in the potential function is only due to I_i being replaced by I'_i and I''_i. Since the densities of I'_i and I''_i are identical, these intervals can be merged as far as the potential function is concerned. Hence the change in the potential function is again given by Equation 1 with x replaced by x', which we know in non-positive.

We can repeat this process until we have used up all of the x work in the arriving job. ■

2.2. A More Competitive Online Algorithm for Large α

We first note that the competitive ratio of every algorithm is at least $O(1)^\alpha$, then give some essential definitions, and then obtain an $O(1)^\alpha$ competitive algorithm.

Lemma 4 *The competitive ratio, with respect to minimizing total energy used, of any randomized online algorithm is* $\Omega((4/3)^\alpha)$.

Proof: We use Yao's Minimax Theorem. Suppose a job of size 1 with deadline 1 arrives at time $t = 0$. With probability 1/2, at time $t = 1/2$, the adversary gives another job of size 1 and deadline 1. Clearly, the offline cost in the case of two jobs is 2^α and 1 if there is a single job. Let us consider how much work that online algorithm does by time $1/2$. If this work is at least $2/3$, then online has spent at least $1/2 \cdot (4/3)^\alpha$ energy already and with probability 1/2 the offline cost is at most 1. If online works less than $2/3$ during the first half, then with probability 1/2, it has to finish at least $4/3$ work in the second half, and hence spend at least $1/2 \cdot (8/3)^\alpha$ energy, whereas the offline cost is 2^α. ■

We assume without loss of generality that all release times and deadlines are integers. Let $u(t_1, t_2)$ denote the work that has release time $\ge t_1$ and has deadline $\le t_2$. For notational convenience, we let $u(t_1, t_2) = u(0, t_2)$ if $t_1 < 0$. Let $u(t, t_1, t_2)$ denote amount of work that has arrived by time t, that has release time $\ge t_1$ and deadline $\le t_2$. Let $y(t)$ denote the rate of optimum offline algorithm YDS at time t. Let $q(t)$ be the maximum over all t_1 and t_2, such that $t_1 < t < t_2$, of $u(t_1, t_2)/(t_2 - t_1)$. Let $p(t)$ be the maximum over all t_1 and t_2, such that $t_1 < t < t_2$, of $u(t, t_1, t_2)/(t_2 - t_1)$. Let $r(t)$ be the maximum over all $t' > t$ of $(u(t, et - (e-1)t', t'))/(e(t' - t))$.

Online Algorithm Description: At time t, work at rate $er(t)$ on the unfinished job with the earliest deadline. Note that $u(t, t_1, t_2)$, $p(t)$ and $r(t)$ may be computed by an online algorithm at time t. We first prove the feasibility of this online algorithm and then analyze its competitiveness.

Theorem 5 *The algorithm completes all the jobs by their deadlines.*

Proof: Assume to reach a contradiction that the online algorithm fails to finish all jobs by some deadline d. We can assume without loss of generality that the online algorithm never idles between time 0 and time d; otherwise we can consider the instance containing only those jobs released after the last idle time. We can also assume that until time d the online algorithm only works on jobs with deadline at most d, since by the earliest deadline first policy, jobs with later deadlines don't affect earlier deadlines. Thus the work done by the online algorithm during $[0, d]$ must be strictly less than $u(0, d)$. The work done by the online algorithm during the time period $[0, d]$ is $\sum_{t \le d} er(t) \ge \sum_{t \le d} u(t, et - (e-1)d, d)/(d-t)$. This inequality follows since we choosing the particular value of t', that is $t' = d$, in the definition of $r(t)$. Let $b(x)$ denote the amount of work that arrives at time x and has deadline $\le d$. Then $\sum_{t \le d} u(t, et - (e-1)d, d)/(d-t) = \sum_{x=0}^{d} \sum_{t=x}^{(x+(e-1)d)/e} b(x)/(d-t) = \sum_{x=0}^{d} b(x) \ln e = u(0, d)$. The first equality follows since for each x, $b(x)$ contributes to $u(t, et - (e-1)d, d)/(d-t)$ if and only if $t \in [x, (x + (e-1)d)/e]$. Thus we have a contradiction to the hypothesis that the online algorithm does less than $u(0, d)$ work before time d. ■

Theorem 6 *The online algorithm is $2(\frac{\alpha}{\alpha-1})^{\alpha} e^{\alpha}$ competitive.*

Proof: Note that $r(t) \le p(t) \le q(t)$. $r(t) \le p(t)$ since $r(t)$ is a special case of $p(t)$ where t_1 is constrained to be a function of t_2. Also $p(t)$ is a lower bound on $q(t)$ based on the input seen by time t. Since $r(t) \le q(t)$, the energy spent by the algorithm is no more than $e^{\alpha} \sum_t q(t)^{\alpha}$. Thus to finish the proof our goal is now to show that $\sum_t q(t)^{\alpha} \le 2(\frac{\alpha}{\alpha-1})^{\alpha} \sum_t y(t)^{\alpha}$.

We now create a new instance where at time t, exactly $y(t)$ units of work arrives with deadline $t+1$. The energy used by YDS does not change. The value of $q(t)$ can only increase because for any t_1, t_2, we trivially have that $u(t_1, t_2) \le \sum_{t=t_1}^{t_2} y(t)$. Thus is sufficient to bound the $q(t)$'s on this new instance. Note that $q(t)$ can be written as the maximum over all t_1 and t_2, such that $t_1 < t < t_2$, of $\sum_{t=t_1}^{t_2} y(t)/(t_2 - t_1)$. Let $l(t)$ be the maximum over all t_1, such that $t_1 < t$, of $\sum_{x=t_1}^{t} y(x)/(t - t_1)$. Similarly, let $r(t)$ be the maximum over all t_2, such that $t \le t_2$, of $\sum_{x=t}^{t_2} y(x)/(t_2 - t)$. Clearly, $q(t) \le \max(l(t), r(t))$, and hence $q(t)^{\alpha} \le l(t)^{\alpha} + r(t)^{\alpha}$. Thus it is sufficient to show that both $\sum_{t=-\infty}^{\infty} l(t)^{\alpha}$ and $\sum_{t=-\infty}^{\infty} r(t)^{\alpha}$ are at most $(\frac{\alpha}{\alpha-1})^{\alpha} \sum_{t>0} y(t)^{\alpha}$.

Note that $l(t)$ is non-zero only for $t > 0$. Thus let us first consider $\sum_{t=0}^{\infty} l(t)^{\alpha}$. We use the following fact that was first proved by Hardy and Littlewood in [6] and later simplified by Gabriel [4]. It can also be found in [7, Theorem 393 and 394].

Fact: If $y(1), \ldots, y(t)$ are arbitrary non-negative integers, let $l(t)$ be the maximum over all v, such that $1 \le v \le t$, of $\sum_{k=v}^{t} y(k)/(t - v + 1)$. Let $s(y)$ be a positive increasing function of y. Let $\bar{y}(1), \ldots \bar{y}(t)$ denote the sequence of $y(i)$ in decreasing sorted order. Then $\sum_{k=1}^{t} s(l(k)) \le \sum_{k=1}^{t} s(\sum_{j=1}^{k} \bar{y}(j)/k)$.

Applying this fact to our case, with $s(y) = y^{\alpha}$, we can conclude that $\sum_{t=0}^{\infty} l(t)^{\alpha}$ is maximized if for all i, $y(i) \ge y(i+1)$. In this case, $l(i) = \sum_{k=1}^{i} y(k)/i$. Thus, $\sum_{t=0}^{\infty} l(t)^{\alpha} \le \sum_{t=0}^{\infty} \left(\sum_{i=1}^{t} y(i)/t\right)^{\alpha}$. Hardy then showed [5] as a special case of Hilbert's theorem (see [7, Theorem 326]) that $\sum_{t=0}^{\infty} \left(\sum_{i=1}^{t} y(i)/t\right)^{\alpha} \le (\frac{\alpha}{\alpha-1})^{\alpha} \sum_t y(t)^{\alpha}$. Note that in these inequalities the $y(i)$'s may be arbitrary, and all that is required of α is that $\alpha > 1$. We our thus done with our analysis of $l(t)$. An identical analysis shows that $\sum_{t=-\infty}^{\infty} r(t)^{\alpha}$ is at most $(\frac{\alpha}{\alpha-1})^{\alpha} \sum y(i)^{\alpha}$. ■

Our online algorithm is e-competitive with respect to the maximum speed since the minimum possible maximum speed for a feasible schedule is at least $\max_t q(t)$.

Lemma 7 *For every deterministic online algorithm A that maintains deadline feasibility, there is some input I that causes A at some time to run e times faster than YDS would ever run on I.*

Proof: The adversary works at a rate $a(t) = \frac{-1}{(\ln x)(1-t)}$ from time 0 to time $1 - x$. We'll look at the limit of the resulting instances as x goes to 0. The work that is released between time f and time g is $\int_f^g \frac{-1}{(\ln x)(1-t)} dt = \frac{1}{\ln x} \ln \frac{1-g}{1-f}$. In particular the work that is released before time $1 - x$ is 1. At time t, the intensity of the interval $[k, 1]$ is then the work released between time k and time t divided by the length of the interval $[k, 1]$. That is, the intensity of the interval $[k, 1]$ is $\frac{1}{(1-k)\ln x} \ln \frac{1-t}{1-k}$. We now want to maximize this intensity for $k \in [0, t]$. Differentiating with respected to k, setting this equal to 0, and solving for $1-k$, gives $1 - k = e(1-t)$, or $k = et - (e-1)$. Note that $et - (e-1) \ge 0$ if and only if $k \ge \frac{e-1}{e}$. Otherwise, for $t \le \frac{e-1}{e}$, the intensity is maximized with $k = 0$.

Thus between time 0 and time $\frac{e-1}{e}$ online can not work at a faster rate than $c\frac{\ln(1-t)}{\ln x}$. If is does so, then the adversary stops bringing work and the online loses. And between time $\frac{e-1}{e}$ and time $1 - x$ online can not work faster than $\frac{-1}{e(1-t)\ln x}$, which we get by plugging $1 - k = e(1-t)$ back into the formula for intensity. If is does so, then the adversary stops bringing work and the online loses. During the time period $1 - x$ to 1, online can not work faster than c times the intensity for the input. The intensity for this input is equal to the apparent intensity for online at time $1 - x$. Thus the intensity for the input is $\frac{-1}{ex \ln x}$.

Thus the most work that online can get done is $c \int_0^{(e-1)/e} \frac{\ln(1-t)}{\ln x} + c \int_{(e-1)/e}^{1-x} \frac{-1}{e(1-t)\ln x} + c \int_{1-x}^{1} \frac{-1}{ex \ln x}$. This work better be at least 1. Solving the integrals gives: $\frac{c}{\ln x}(\frac{2}{e} - 1) + \frac{c}{e \ln x}(\ln x + 1) + \frac{-c}{e \ln x} \ge 1$, or equivalently $\frac{c}{\ln x}(\frac{2}{e} - 1 + \frac{\ln x}{e}) \ge 1$. As x goes to 0, $\ln x$ is a huge negative number, thus the $\frac{3}{e} - 1$ becomes irrelevant. What is important is that $c \ge e$. ■

3. Minimizing the Maximum Temperature

We consider dynamic speed scaling to minimize the maximum temperature. We show how to solve this problem to arbitrary precision in polynomial time (ignoring numerical stability issues) using the Ellipsoid algorithm. For basic information on the use of the Ellipsoid algorithm to solve convex problems see [10]. We assume a constant T_{max} that is the thermal threshold for the device. The problem is then to determine if there is a schedule that is feasible, and maintains the invariant that the temperature stays below T_{max}. By binary search, we can then solve the problem of minimizing T_{max}. To simplify notation we assume that the cube root rule holds throughout this section, that is $\alpha = 3$. The extension to arbitrary α is straight-forward.

We first express the problem as a convex program. We break time into intervals $t_0, \ldots t_m$ at release dates and deadlines of the jobs. We introduce a variable T_i that represents $T(t_i)$, the temperature at time t_i. Let $J(i)$ be the jobs j that can feasibly be executed during time $[t_i, t_{i+1}]$, that is, $r_j < t_{i+1}$ and $d_j > t_i$. We introduce a variable $W_{i,j}$, for $j \in J(i)$, that represents the work done on job j during time $[t_i, t_{i+1}]$. Let $MaxW(x, y, X, Y)$ be the maximum work that can be done starting at time x at temperature X and ending at time y at temperature Y subject to the temperature constraint $T \leq T_{max}$ throughout the interval $[x, y]$. Let $MaxT(x, y, X, y)$ be a corresponding temperature curve. Let $UMaxW(x, y, X, Y)$ and $UMaxT(x, y, X, Y)$ be similarly defined for except that there is no maximum temperature constraint. We can then express the temperature problem as the mathematical program CP:

$$p_j \leq \sum_{i: j \in J(i)} W_{i,j} \qquad 1 \leq j \leq n \qquad (2)$$

$$\sum_{j \in J(i)} W_{i,j} \leq MaxW(t_i, t_{i+1}, T_i, T_{i+1}) \qquad 1 \leq i \leq m-1 \qquad (3)$$

$$0 \leq T_i \qquad 1 \leq i \leq m \qquad (4)$$

$$0 \leq W_{i,j} \qquad 1 \leq i \leq m, 1 \leq j \leq n \qquad (5)$$

To see that the feasible region is convex, let $\tilde{T}$, and $\hat{T}$ be the temperature curves corresponding to two feasible solutions to this problem. Let $\bar{T} = (\tilde{T} + \hat{T})/2$. The power curve corresponding to $\bar{T}$ is $\bar{P} = \left((\bar{T}' + b\bar{T})/a\right)^{1/3} = \left((\tilde{T}' + \hat{T}' + b\tilde{T} + b\hat{T})/(2a)\right)^{1/3}$. Then since $x^{1/3}$ is a concave function $\bar{P} \geq (\tilde{P} + \hat{P})/2$. Thus the average of the two underlying feasible solutions is feasible.

To apply the Ellipsoid algorithm one needs to give a procedure to determine whether an arbitrary point is feasible, and if not, to determine a separating hyperplane. To accomplish this, one needs to better understand the function $MaxW(t_i, t_{i+1}, T_i, T_{i+1})$. And to understand the unconstrained problem $UMaxW_i(t_i, t_{i+1}, T_i, T_{i+1})$.

3.1. The Unconstrained Maximum Work Problem

We consider the unconstrained problem $UMaxW(t_0, t_1, T_0, T_1)$, where the times and temperatures are arbitrary. For convenience, we assume $t_0 = 0$. $UMaxT$ is a function T of time t such that quantity $\int_0^{t_1} P^{1/3} dt = \int_0^{t_1} \left(\frac{T'+bT}{a}\right)^{1/3} dt$ is maximized subject to the constraints that $P \geq 0, T(0) = T_0$ and $T(t_1) = T_1$. Here T' is the derivative of T with respect to time t. This problem falls under the rubric of calculus of variations [14]. Let F be the functional $((T' + bT)/a)^{1/3}$. Let $F_T = \frac{b}{3a^{1/3}}(T' + bT)^{-2/3}$ be the partial of F with respect to T, and $F_{T'} = \frac{1}{3a^{1/3}}(T' + bT)^{-2/3}$ be the partial of F with respect to T'. Since F has continuous first and second partial derivatives with respect to all arguments, any weak extremum T must satisfy the Euler-Lagrange equation $F_T - \frac{d}{dt}F_{T'} = 0$. We call a function T that satisfies such an equation an Euler curve. The term $\frac{d}{dt}F_{T'} = \frac{-2}{9a^{1/3}}(T' + bT)^{-5/3}(T'' + bT')$. Thus the Euler-Lagrange equation evaluates to $b(T' + bT)^{-2/3} + \frac{2}{3}(T' + bT)^{-5/3}(T'' + bT') = 0$. We multiply by $(T' + bT)^{5/3}$ to give $b(T' + bT) + \frac{2}{3}(T'' + bT') = 0$, or $3b^2T + 5bT' + 2T'' = 0$. Using the standard Laplace transform technique we get that the solution is $T = ce^{-bt} + de^{-3bt/2}$, where the constants c and d are determined by the boundary conditions.

We now compute the values of c and d. Setting $t = 0$ and $T = T_0$ we get that $c + d = T_0$. Similarly setting $t = t_1$ and $T = T_1$ we get that $ce^{-bt_1} + de^{-3bt_1/2} = T_1$. Setting $c = T_0 - d$, we get that $(T_0 - d)e^{-bt_1} + de^{-3bt_1/2} = T_1$. Hence, $d = (T_0 e^{-bt_1} - T_1)/(e^{-bt_1} - e^{-3bt_1/2})$. Note that since $P \geq 0, T_1 \geq T_0 e^{-bt_1}$, and thus $d \leq 0$. By direct differentiation, $T' = -bce^{-bt} - 3bd/2e^{-3bt/2}$, or equivalently $T' = -be^{-bt}(T_0 - d + 3d/2e^{-bt/2})$.

We turn our attention back to evaluating our equation for energy used. Evaluating $T' + bT$, the term which depends on ce^{-bt} cancels out, and we just get $-d/2be^{-3bt/2}$. Hence $UMaxW$ is $\int_0^{t_1} a^{-1/3}(-db/2)^{1/3}e^{-bt/2}dt$, which evaluates to $(-4d/ab^2)^{1/3}(1 - e^{-bt_1/2})$. We now turn our attention to computing the maximum temperature reached on the Euler curve.

Lemma 8 *For any Euler curve T, there exists a unique time t_x such that $T' \geq 0$ for $t \in [0, t_x]$ and $T' \leq 0$ for $t \in [t_x, t_1]$.*

Proof: Note that $e^{bt}T'/b = -T_0 + d - 3de^{-bt/2}/2$, which is a non-increasing function of t since d is non-positive. This precludes T' ever going from positive to negative. ■

Corollary 9 *If $T'(0) \leq 0$, a maxima of the Euler curve T is at $t = 0$. If $T'(t_1) \geq 0$, a maxima of the Euler curve T is at $t = t_1$. If $T'(0) \geq 0$ and $T'(t_1) \leq 0$, then the maxima of the Euler curve T is at some unique intermediate point t_x where $T'(t_x) = 0$.*

Lemma 10 *Consider the class of Euler curves between $(0, T_0)$ and (t_1, T_1). If an Euler curve T is such that $T'(0) \geq 0$ and $T'(t_1) \leq 0$, then the maximum temperature that the Euler curve reaches is $\frac{4c^3}{27d^2}$. Furthermore, this maximum temperature is an increasing function of t_1.*

Proof: At the maximum temperature point we know that $T' = 0$. Hence, $ce^{-bt} = -\frac{3}{2}de^{-3/2bt}$, or equivalently, $\frac{-2c}{3d} = e^{-bt/2}$. Plugging this into the equation for the temperature we get that the maximum temperature is $-d/2e^{-3/2bt} = \frac{4c^3}{27d^2}$.

We now consider the second statement of the lemma. The condition that $T'(0) \geq 0$ implies that $-c - \frac{3}{2}d \geq 0$, and hence $2T_0 + d \leq 0$. The condition that $T'(t_1) \leq 0$ implies that $-ce^{-bt_1} - \frac{3}{2}de^{-3bt_1/2} \leq 0$, or equivalently, $(d - T_0) - \frac{3}{2}de^{-bt_1/2} \leq 0$, or equivalently, $(T_0e^{-3bt_1/2} - T_1)(e^{-bt_1} - e^{-3bt_1/2}) - \frac{3}{2}(T_0e^{-3bt_1/2} - T_1e^{-bt_1/2})(e^{-bt_1} - e^{-3bt_1/2}) \leq 0$. This implies that $2T_1 - 3T_1e^{-bt_1/2} + T_0e^{-3bt_1/2} \geq 0$

We now differentiate the maximum temperature, or more simply c^3/d^2, with respect to t_1. The derivative of c^3/d^2 is $3c^2c'/d^2 - 2c^3d'/d^3$. Since $c' = -d'$, this can be written as $-\frac{c^2d'}{d^3}(3d + 2c) = -\frac{c^2d'}{d^3}(2T_0 + d)$. We want to show that this derivative is positive. Since $d \leq 0$, it suffices to show that $d'(2T_0 + d) \geq 0$. Since we know that $2T_0 + d \leq 0$, it suffices to show that $d' \leq 0$. By direct computation, d' is $-(T_0e^{-3bt_1/2} - 3T_1e^{-bt_1/2} + 2T_1)(2e^{bt}(e^{-bt_1} - e^{-3bt_1/2})^2)$. Since we know that $2T_1 - 3T_1e^{-bt_1/2} + T_0e^{-3bt_1/2} \geq 0$, and the denominator is clearly positive, it then follows that $d' \leq 0$. And we can conclude that the maximum temperature is an increasing function of t_1. ■

3.2. The Temperature Constrained Maximum Work Problem

We now turn our attention to $MaxW(t_0 = 0, t_1, T_0, T_1)$, that is we now assume the existence of a temperature constraint $T \leq T_{max}$. If one adds an inequality constraint to an unconstrained problem such as ours, the resulting extremum curve can be decomposed into subcurves, where either it is the case the the subcurve is an Euler curve, it is the case that the inequality constraint holds with equality on the subcurve [14]. An immediate consequence of the following lemma is that the portion of $MaxT$, where $MaxT = T_{max}$, is a single line segment.

Lemma 11 *Consider two points $(0, T_x)$ and (t_1, T_x), Then let L_1 denote the constant temperature curve that connects $(0, T_x)$ and (t_1, T_x). Let L_2 denote some temperature curve such $L_2(t) \leq L_1(t)$ for $t \in [0, t_1]$. Then L_1 does at least as much work as L_2.*

Proof: Consider a horizontal line of temperature T_1 of infinitesimally small width (call it dT) where $T_1 < T$, such that it intersects L_2 in at least two places. Let X_1 and X_2 be two consecutive times when the line T_1 and L_2 intersect, where at X_1, L_2 transitions from being above T_1 to being below T_1. Hence, at X_2, L_2 transitions from being below T_1 to being above T_1. Let dt_1 and dt_2 be infinitesimal periods of time around X_1 and X_2. Let P_1 and P_2 denote the power applied during X_1 and X_2 (we are assuming that the widths of X_1 and X_2 are infinitesimally small, so we can think of the power as a constant). We know that since L_2 is decreasing at X_1 it is the case that $\frac{-dT}{dt_1} = L_2'(X_1) = aP_1 - bT_1$. Similarly, $\frac{dT}{dt_2} = L_2'(X_2) = aP_2 - bT_1$. It will be sufficient to show that the work done by L_2 during during dt_1 and dt_2 is at most the work done by L_1 during dt_1 and dt_2. The work done by L_2 during these intervals, is $P_1^{1/3}dt_1 + P_2^{1/3}dt_2$, The work done by L_1 during these intervals, $(bT/a)^{1/3}(dt_1 + dt_2)$, which is at least $(bT_1/a)^{1/3}(dt_1 + dt_2)$. Thus it is sufficient to show that $[P_1^{1/3} - ((bT_1)/a)^{1/3}]dt_1 + [P_2^{1/3} - ((bT_1/a)^{1/3}]dt_2 \leq 0$. By dividing by $(bT_1/a)^{1/3}$, this is equivalent to $[(aP_1/bT_1)^{1/3} - 1]dt_1 + [(aP_2/bT_1)^{1/3} - 1]dt_2 \leq 0$.

Let $x = aP_1/bT_1 - 1$ and $y = aP_2/bT_1 - 1$. Note that $x = L_2'(X_1)/(bT_1)$ and $y = L_2'(X_1)/(bT_1)$. Observe that since $P_1 \geq 0$ and temperature in decreasing in the region X_1, it is the case that $0 \leq x + 1 < 1$, or equivalently $-1 \leq x < 0$. Since L_2 is increasing during X_2, $y > 0$. Further observe that $xdt_1 + ydt_2 = 0$. The expression that we need to prove can now be rewritten as $[(x+1)^{1/3} - 1]dt_1 + [(y+1)^{1/3} - 1]dt_2 \leq 0$. Now $dt_2 = -xdt_1/y$, so we need to show $y[(x+1)^{1/3} - 1] - x[(y+1)^{1/3} - 1] \leq 0$. Call $u = (x+1)^{1/3}$ and $v = (y+1)^{1/3}$, so we need to show $(v^3 - 1)(u - 1) - (u^3 - 1)(v - 1) \leq 0$. Note that $0 \leq u \leq 1$ and $v > 1$. If $u = 1$ then it this is trivially true, so assume $u < 1$. Then dividing through by $(v - 1)$ and $(u - 1)$, it is sufficient to show that $(v^2 + v + 1) \geq (u^2 + u + 1)$, which holds since $u \leq v$. ■

We now know that either $MaxT = UMaxT$, or the $MaxT$ consists of three parts: an Euler curve up to T_{max}, a line segment at T_{max} and an Euler curve down to T_1. We will now consider the case that $MaxT \neq UMaxT$ and consider the different parts of this composite curve. In particular we want to compute the times γ and $T_1 - \beta$, such that $MaxT(t) = T_{max}$ exactly when $t \in [\gamma, T_1 - \beta]$. First we define these times, and then we show that they have this desired property.

Consider the class of Euler curves T that go from $(0, T_0)$ to (t_1, T_{max}) for different possible t_1's. A necessary and sufficient condition for T to have a maximum at t_1 is that $T'(t_1) \geq 0$. If $T'(t_1) < 0$, there is some time earlier than t_1 with temperature greater than T_{max}. By direct computation $T'(t_1)$ is $-be^{-bt_1}(c + 3de^{-bt_1/2}/2)$. To show that $T'(t_1) \geq 0$ is is sufficient to show that $c + 3de^{-bt_1/2}/2 \leq 0$. This is equivalent to showing that $T_{max} + T_0e^{-3bt_1/2}/2 - 3T_{max}e^{-bt_1/2}/2 \leq 0$. We claim that the left had side of this equation is increasing in t_1. The derivative of the left hand side as a function of t_1 is $-3bT_0e^{-3bt_1/2}/4 + 3bT_{max}e^{-bt_1/2}/4$ which is always positive. Since $T'(0) \leq 0$, there is a there is a unique value of t_1 such that $T'(t_1) = 0$. We denote this value of t_1 by γ. That is, γ is the solution to the equation to the equation $T_0e^{-3b\gamma/2} + 2T_{max} - 3T_{max}e^{-b\gamma/2} = 0$. Note that for $t_1 \leq \gamma$, the temperature is never above T_{max}.

Consider the class of Euler curves T that go from $(0, T_{max})$ to (t_1, T_1) for different possible t_1's. If $T'(0) > 0$ then the Euler curve exceeds T_{max} just to the right of 0. So let us study the condition that $T'(0) \leq 0$. By direct computation $T'(0)$ is $-b(c + 3d/2) = -b(T_{max} + d/2)$. Evaluating this we get that $T'(0) = -b(T_{max}e^{-bt_1} - T_{max}e^{-3bt_1/2} - T_1/2)/2$. The partial derivative of $T'(0)$ with respect to t_1 is $b^2T_{max}(3e^{-bt_1}/2 - 3e^{-3bt_1/2}/2)$, which is always positive. Since $T'(0) \leq 0$, there is a unique t_1 such that $T'(t_1) = 0$. We denote this value of t_1 by β. That is, β is the solution to the equation $T_{max}e^{-3b\beta/2} + 2T_1 - 3T_1e^{-b\beta/2} = 0$. Note that for $t_1 \leq \beta$, the temperature is never above T_{max}.

Theorem 12 *We consider the curve $MaxT(0, T_0, t_1, T_1)$. Let γ and be β defined as above. If $t_1 \leq \gamma + \beta$, then $MaxT = UMaxT$. If $t_1 > \gamma + \beta$, then the curve $MaxT$ travels along the Euler curve from $(0, T_0)$ to (γ, T_{max}), then stays at T_{max} until time $t_1 - \beta$, and finally travels along the Euler curve from $(t_1 - \beta, T_{max})$ to (t_1, T_1).*

Proof: First note that if $t_1 = \gamma + \beta$, then the curve as described above is the Euler curve, because all points satisfy the Euler condition. The maxima of this curve is T_{max}. By Lemma 10 we know that the maximum temperature that $UMaxT$ reaches is a increasing function of t_1. Hence, if $t_1 < \gamma + \beta$ then $UMaxT < T_{max}$ and $MaxT = UMaxT$.

Now consider the case that $t_1 > \gamma + \beta$. We know that the loci of points on $MaxT$ with temperature T_{max} forms a line segment L than runs from time t_x to t_y. Note that by the definition of γ and β, it is the case that $t_x < t_y$. We claim that it must be the case that $t_x = \gamma$. If $t_x > \gamma$ then you would have a contradiction to the feasibility of $MaxT$, since by definition, γ is the latest time where Euler curve doesn't violate the T_{max} temperature constraint. If $t_x < \gamma$ one gets a contradiction to optimality of $MaxT$. This is because one could replace the portion of $MaxT$, from time 0 to a time slightly larger than t_x, with an unconstrained Euler curve time that would finish more work. An identical argument implies that $t_b = t_1 - \beta$. ∎

3.3. Computing a Separating Hyperplane

We now are ready to address of the problem of determining whether a point is feasible in our convex programming formula of the problem, and if not, finding a separating hyperplane. Consider an arbitrary point where each T_i takes the value $\hat{T}_i$, and each $W_{i,j}$ takes the value $\hat{W}_{i,j}$. The only problematic constraints are the $MaxW$ constraints. We thus focus on these constraints for the remainder of this subsection. Consider the ith such constraint. Given the values of t_i, t_{i+1}, $\hat{T}_i$ and $\hat{T}_{i+1}$ we can compute the values γ and β as defined in the last subsection by binary search using the defining equations.

Consider the case that $t_{i+1} - t_i \leq \gamma + \beta$. Let $f = t_{i+1} - t_i$. Then we know that maximum temperature constraint is not relevant here. From our earlier results we then know that the maximum work that can be completed is $UMaxW = (-4d/ab^2)^{1/3}(1 - e^{-bf/2})$, where $d = (T_ie^{-bf} - T_{i+1})/(e^{-bf} - e^{-3bf/2})$. We can then determine whether the the ith $MaxW$ constraint is violated. If this constraint is violated, to compute a separating hyperplane let G be a function of T_i, T_{i+1}, and $W_{i,j}$ for $j \in J(i)$ defined as $\sum_{j \in J(i)} W_{i,j} - UMaxW(t_i, t_{i+1}, T_i, T_{i+1})$. The ith $MaxW$ constraint is then equivalent to $G \leq 0$. A separating hyperplane is then the plane whose normal is the gradient of G evaluated at the current point. Note that we can easily differentiate G with respect to all of its variables.

Now consider the case that $t_{i+1} - t_i \geq \gamma + \beta$. Now $MaxW$ is given by the work done by the Euler curve between (t_i, T_i) and $(t_i + \gamma, T_{max})$, plus the work done on the constant temperature curve at T_{max} between time $(t_i + \gamma)$ and time $(t_{i+1} - \beta)$, plus the work done by the Euler curve from $(t_{i+1} - \beta, T_{max})$ to (t_{i+1}, T_{i+1}). Thus $MaxW$ is

$$\left(\tfrac{-4d_1}{ab^2}\right)^{1/3}(1 - e^{-b\gamma/2}) + \left(\tfrac{-4d_2}{ab^2}\right)^{1/3}(1 - e^{-b\beta/2}) \\ +(t_{i+1} - t_i - \gamma - \beta)\left(\tfrac{bT_{max}}{a}\right)^{1/3}$$

where $d_1 = (T_ie^{-b\gamma} - T_{max})/(e^{-b\gamma} - e^{-3b\gamma/2})$, and $d_2 = (T_{max}e^{-b\beta} - T_{i+1})/(e^{-b\beta} - e^{-3b\beta/2})$. We can determine whether this $MaxW$ constraint is violated. If this constraint is violated, to compute a separating hyperplane let G be a function of T_i, T_{i+1}, and $W_{i,j}$ for $j \in J(i)$ defined as $\sum_{j \in J(i)} W_{i,j} - MaxW(t_i, t_{i+1}, T_i, T_{i+1})$. The ith $MaxW$ constraint is then equivalent to $G \leq 0$. A separating hyperplane is then the plane whose normal is the gradient of G evaluated at the current point. The only slightly tricky part of computing the gradient is differentiating γ with respect to T_i, and differentiating β with respect to T_{i+1}. This can be accomplished using the equations that define γ and β. Recall that γ is the solution to the equation $T_i e^{-3b\gamma/2} + 2T_{max} - 3T_{max}e^{-b\gamma/2} = 0$. Differentiating γ with respect T_i and solving for $d\gamma/dT_i$ yields $d\gamma/dT_i = (-2e^{-b\gamma})/((3b(T_{max} - T_i e^{-b\gamma}))$. Recall that β is the solution to the equation $T_{max}e^{-3b\beta/2} + 2T_{i+1} - 3T_{i+1}e^{-b\beta/2} = 0$. Differentiating β with respect T_{i+1} and solving for $d\beta/dT_{i+1}$ yields $d\beta/dT_{i+1} = (-4 + 6e^{-b\beta/2})/((3b(T_{i+1}e^{-b\beta/2} - T_{max}e^{-3b\beta/2}))$.

Acknowledgements: We would like to thank Mary Jane Irwin, Bruce Childers, Rami Melhem, Daniel Mosse, Patchrawat Uthaisombut, Gerhard Woeginger, Maxim Sviridenko, Sandy Irani, and Cliff Stein for interesting discussions on power aware computing.

References

[1] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *IEEE Syposium on Foundations of Computer Science*, 2004.

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

[3] G. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer, 1997.

[4] R. M. Gabriel. An additional proof of a maximal theorem of hardy and littlewood. *Journal of London Mathematical Society*, 6:163–166, 1931.

[5] G. H. Hardy. Note on a theorem of hilbert. *Math. Zeitschr.*, 6:314–317, 1920.

[6] G. H. Hardy and J. Littlewood. A maximal theorem with function-theoretic applications. *Acta Mathematica*, 54:81–116, 1930.

[7] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.

[8] S. Irani, , R. K. Gupta, and S. Shukla. Algorithms for Power Savings. In *ACM/SIAM Symposium on Discrete Algorithms*, 2003.

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

[10] Y. Nestorov. Introductory lectures on convex programming.

[11] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.

[12] J. E. Sergent and A. Krum. *Thermal Management Handbook*. McGraw-Hill, 1998.

[13] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, pages 2–13, 2003.

[14] D. R. Smith. *Variational Methods in Optimization*. Prentice-Hall, 1974.

[15] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Design Automation Conference*, pages 732–737, 1998.

[16] http://www.usatoday.com/tech/news/2004-05-07-intel-kills-tejas_x.htm.

[17] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science*, page 374, 1995.