Every Deterministic Nonclairvoyant Scheduler has a Suboptimal Load Threshold

Jeff Edmonds[*]

Abstract

We prove a surprising lower bound for resource augmented nonclairvoyant algorithms for scheduling jobs with sublinear nondecreasing speed-up curves on multiple processors with the objective of average response time. Edmonds in STOC99 shows that the algorithm Equi-partition is a $(2+\epsilon)$-speed $\Theta(\frac{1}{\epsilon})$-competitive algorithm. We define its *speed threshold* to be 2 because it is constant competitive when given speed $2+\epsilon$ but not when given speed 2. (Its *load threshold* is the inverse of its speed threshold.) The *optimal* speed threshold is 1 because then the algorithm is constant competitive no matter how little extra resources it is given. Edmonds and Pruhs in SODA09 imply that they have found such an algorithm. (They use the term *scalable.*) We, however, rebut that their algorithm only accomplishes this *nondeterministically.* They prove that for every $\epsilon > 0$, there is an algorithm Alg_ϵ that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive. A problem, however, is that this algorithm Alg_ϵ depends on ϵ. Hence, to have one algorithm it would have to runs Alg_ϵ after nondeterministically guessing the correct ϵ. We prove that like Equi-partition, every fixed deterministic nonclairvoyant algorithm has a suboptimal speed threshold, namely for every (graceful) algorithm Alg, there is a threshold $1+\beta_{\mathrm{Alg}}$ that is $\beta_{\mathrm{Alg}} > 0$ away from being optimal such that the algorithm $\omega(1)$ competitive with speed $1+\beta_{\mathrm{Alg}}$. We go on to prove that choosing an algorithm is a trade off between its performance and the peek load it can handle. Though the SODA09 paper allows us to choose a algorithm with a speed threshold arbitrarily close to one, it comes at a cost of $\frac{1}{\beta_{\mathrm{Alg}}}$ in its competitive ratio, because even when given speed $(1+\beta_{\mathrm{Alg}})+\epsilon$, the competitive ratio is at least $\Omega(\frac{1}{(\beta_{\mathrm{Alg}}+\epsilon)\beta_{\mathrm{Alg}}})$.

In addition to being an interesting result, the proof technique is quite novel. It use Brouwer's fixed point theorem to break the cycle of needing to know which input will be given before one can know what the algorithm will do and needing to know what the algorithm will do before one can know which input to give.

Key words: lower bound, resource augmented competitive ratio, nonclairvoyant online scheduling, multi-processor, average response time, nondeterministic, Brouwer's fixed point theorem

[*]York University, Canada. jeff@cse.yorku.ca. Supported in part by NSERC Canada.

1 Introduction

Computer chip designers agree upon the fact that chips with hundreds to thousands of processors chips will dominate the market in the next decade. The founder of chip maker Tilera asserts that a corollary to Moore's law will be that the number of cores/processors will double every 18 months [?]. Intel's director of microprocessor technology asserts that while processors will get increasingly simple, software will need to evolve more quickly than in the past to catch up [?]. In fact, it is generally agreed that developing software to harness the power of multiple processors is going to be a much more difficult technical challenge than the development of the hardware. In this paper, we consider one such software technical challenge: developing operating system algorithms/policies for scheduling processes with varying degrees of parallelism on a multiprocessor.

We will consider the setting where n processes/jobs arrive to the system over time. Job J_i arrives at time r_i, has a work w_i, and *speedup function* $\Gamma_i(\rho)$ specifying the rate at which work is completed when allocated ρ of the p processors, [?]. The upper bounds consider jobs with multiple phases each with an arbitrary sublinear nondecreasing speedup curve. Our lower bound, on the other hand, only considers *parallelizable* jobs which have $\Gamma_i(\rho) = \rho$ and *sequential* jobs with $\Gamma_i(\rho) = 1$.

An operating system scheduling algorithm Alg generally needs to be *online* in that it does not know what jobs will arrive in the future and *nonclairvoyant* in that it does not know either w_i nor $\Gamma_i(\rho)$. At each point of time, the algorithm specifies the number of processors $\rho_{\langle i,t\rangle}$ that it allocates to each job J_i. If a job J_i completes at time C_i, then its response time is $C_i - r_i$. In this paper we will consider the schedule quality of service metric *total response time*, $F(\text{Alg}(J)) = \sum_{i=1}^{n}(C_i - r_i) = \int_t n_t \delta t$. This (or equivalently average response time) is by far the mostly commonly used schedule quality of service metric. (Because we allow the allocation $\rho_{\langle i,t\rangle}$ to be any real number, we can scale p to one.)

Let us now review the definitions of competitive ratio and resource augmentation. A scheduling algorithm Alg is s-speed c-competitive if $\max_J \frac{F(\text{Alg}_s(J))}{F(\text{Opt}_1(J))} \le c$ where $\text{Alg}_s(J)$ is the algorithm with sp processors and $\text{Opt}_1(J)$ is the optimal schedule when given only p processors [?, ?]. Instead of giving Alg extra speed s, you could equivalently think of giving it reduced load $L = \frac{1}{s}$. We say that algorithm Alg has *speed threshold* $1+\beta_{\text{Alg}}$ and *load threshold* $\frac{1}{1+\beta_{\text{Alg}}}$ if it is a $((1+\beta_{\text{Alg}})+\epsilon)$-speed $\Theta(\frac{1}{\epsilon^{\mathcal{O}(1)}})$-competitive algorithm. The *optimal* speed/load threshold is 1 because then the algorithm is constant competitive no matter how little extra resources it is given. The literature [?, ?, ?] uses the term *scalable* for such an algorithm. To understand the motivation for these definitions consider the sort of quality of service curves that are ubiquitous in server systems. See figure **??**. That is, there is a relatively modest degradation in quality of service as the load increases until one nears some threshold, after which any increase in the load precipitously degrades the quality of service provided by the server. Load, which generally reflects the number of users of the system, is formalized here as follows. Lets say that a job stream J has *load* $L \in [0, 1]$, if $F(\text{Opt}_L(J)) = 1$, namely the stream can be optimally handled with speed L. Note that with this definition, *load* increases with $L \in [0, 1)$ and is unmanageable for load $L = 1$ for any nonclairvoyant speed 1 scheduler. Lemma **??** (in the appendix) proves that the quality of service of an algorithm given load L is equal to its competitive ratio when given speed $s = \frac{1}{L}$. Hence, if the competitive ratio sky rockets with speed below the algorithm's speed threshold $s = 1 + \beta_{\text{Alg}}$, then its quality of service does so with load above $L = \frac{1}{1+\beta_{\text{Alg}}}$.

No nonclairvoyant scheduling algorithm can be $O(1)$-competitive for total response time if compared against the optimal schedule with the same speed [?], even if all the jobs are parallelizable. The intuition is that one can construct adversarial instances where the load is essentially the

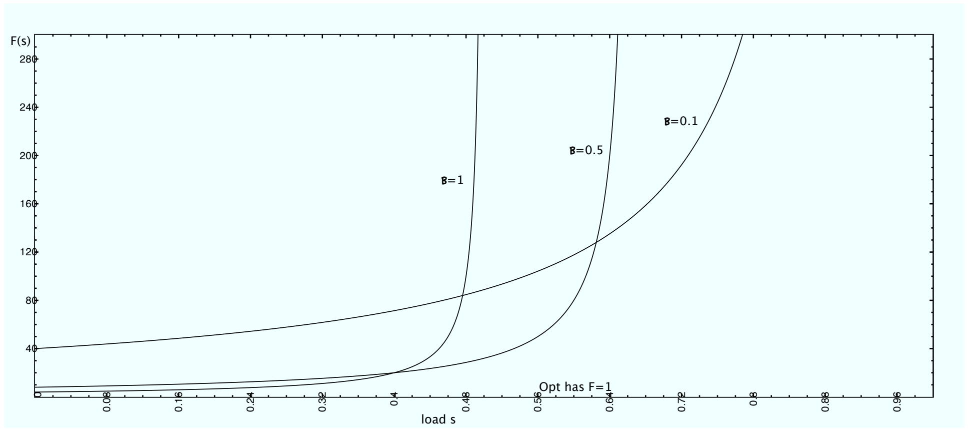

Figure 1: The quality of service curve for the algorithm LAPS_β for $\beta = 1$, 0.5, and 1. The speed threshold for this algorithm is roughly $1+\beta$ and its load threshold $\frac{1}{1+\beta} = 0.5$. 0.67, and 0.91. Note how the quality of service increases with $\frac{1}{\beta}$. (Sorry the load label should be L not s.)

capacity of the system, and there is no time for the nonclairvoyant algorithm to recover from any scheduling mistakes. Theorem **??** uses basically the same instance.

The nonclairvoyant algorithm Shortest Elapsed Time First (SETF) on parallelizable jobs is $(1+\epsilon)$-speed $\Theta(\frac{1}{\epsilon})$-competitive and hence has the optimal speed threshold [**?**] when all the jobs are fully parallelizable. However, it performs very poorly when there are sequential jobs. SETF shares the processor equally among all processes that have been processed the least to date. Intuitively, SETF gives priority to more recently arriving jobs, until they have been processed as much as older jobs, at which point all jobs are given equal priority. The process scheduling algorithm used by most standard operating systems, e.g. Unix, essentially schedules jobs in way that is consistent with this intuition.

The most obvious scheduling algorithm in the multiprocessor setting is Equi-partition (Equi), which splits the processors evenly among all processes. Equi is analogous to the Round Robin or Processor Sharing algorithm in the single processor setting. It is shown in [**?**] that Equi is $(2+\epsilon)$-speed $(\frac{2s}{\epsilon})$-competitive. It is also known that, even in the case of only parallelizable jobs, speed at least $2+\epsilon$ is required in order for Equi to be $O(1)$-competitive for total response time [**?**].

The paper [**?**] introduces a new algorithm call LAPS_β (Latest Arrival Processor Sharing). It is parameterized by a real $\beta \in (0,1]$. It partitions the processors equally among the $\lceil \beta n_t \rceil$ jobs with the latest arrival times, where n_t is the number of jobs alive at time t. Note that LAPS_β is a generalization of Equi since LAPS_1 is identical to Equi. But as β decreases, LAPS_β, in a manner reminiscent of SETF, favors more recently released jobs. They prove that LAPS_β with $s = (1+\beta+\epsilon)$ processors is $\left(\frac{4s}{\beta\epsilon}\right)$-competitive.

1.1 Our Results

We rebut the suggestion in [**?**] that their algorithm LAPS_β is *scalable*, i.e. has optimal speed threshold 1. We show that their algorithm only accomplishes this *nondeterministically.*

The Desired Result: $\exists \mathrm{Alg}\ \forall\epsilon\ \frac{F(\mathrm{Alg}_1(J))}{F(\mathrm{Opt}_{1/(1+\epsilon)}(J))} = \frac{1}{\epsilon^{\mathcal{O}(1)}}$

Their Upper Bound: $\forall\epsilon\ \exists \mathrm{Alg}_\epsilon\ \frac{F(\mathrm{Alg}_1(J))}{F(\mathrm{Opt}_{1/(1+\epsilon)}(J))} = \mathcal{O}(\frac{1}{\epsilon^2})$, namely $\mathrm{Alg}_\epsilon = \mathrm{LAPS}_\beta$ with $\beta = \frac{\epsilon}{2}$.

Our Lower Bound: $\forall \text{Alg}\ \exists \epsilon\ \frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} = \omega(1)$. For example, to make $\text{LAPS}_{\langle \beta, 1+\epsilon\rangle}$ noncompetitive, set $\epsilon = \frac{\beta}{2}$.

For them to have one scalable algorithm, it would have to runs Alg_ϵ after nondeterministically guessing the correct ϵ. In contrast, our result proves that every deterministic non-clairvoyant algorithm Alg has a speed threshold $1+\beta_{\text{Alg}}$ that is $\beta_{\text{Alg}} > 0$ away from being optimal. We go on to prove that choosing an algorithm is a trade off between its performance and the peek load it can handle. See Figure **??**. Equi has the best performance, but it only can handle "half" load. LAPS_β with a small β can handle almost full load, but its performance degrades with $\frac{1}{\beta}$.

Theorem 1. *For all* graceful deterministic sufficiently non-clairvoyant *algorithms* Alg*, there exists a speed threshold* $1+\beta_{\text{Alg}}$ *that is* $\beta_{\text{Alg}} > 0$ *away from being optimal so that with speed* $(1+\beta_{\text{Alg}})+\epsilon$*, the algorithm is* $\Omega(\frac{1}{(\beta_{\text{Alg}}+\epsilon)\beta_{\text{Alg}}})$ *competitive and with speed* $1+\beta_{\text{Alg}}$*, the algorithm is* $\omega(1)$ *competitive.*

This lower bound is completely tight with the upper bounds for LAPS_β. In fact, we prove that (triangle) LAPS_β is optimal. The restrictions that the algorithm is graceful is defined later and is believe to be minor.

1.2 Related Results

For the objective of total response time on a single processor, the competitive ratio of every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\ln n)$ [**?**]. There is a randomized algorithm, Randomized Multi-Level Feedback Queues, that is $O(\ln n)$-competitive against an oblivious adversary [**?**, **?**]. The online clairvoyant algorithm Shortest Remaining Processing time is optimal for total response time. The competitive analysis of SETF_s for single processor scheduling was improved for cases when $s \gg 1$ in [**?**].

Variations of Equipartition are built into many technologies. For example, the congestion control protocol in the TCP Internet protocol essentially uses Equipartition to balance bandwidth to TCP connections through a bottleneck router. Extensions of the analysis of Equi in [**?**] to analyzing TCP can be found in [**?**, **?**]. Other extensions to the analysis of Equi in [**?**] for related scheduling problems can found in [**?**, **?**, **?**]. An application in another area is that [**?**, **?**] reduces the model with sequential jobs to the model in which a page can be simultaneously broadcast to many users.

Another area of increased interest is speed scaling. In this model, the scheduler gets to decide how much extra resources it gets but must pay for the *energy* that it consumes. [**?**] shows that a version of LAPS_β is $\Theta(\alpha^2)$-competitive when there is only one processor and the objective is average response time plus energy and the energy consumed is $P(s_t) = (s_t)^\alpha$ when the processor is at speed s_t. [**?**] generalizes it to include sublinear nondecreasing speed-up curves on multiple processors. It is $\Theta(\alpha^2)$-competitive when the energy consumed is $P(p_t) = (p_t)^\alpha$ when the scheduler allocates p_t unit speed processors. It is $\Theta(\log(p))$-competitive when the energy consumed is $P(\vec{s}_t) = \sum_{i\in[p]}(s_{\langle i,t\rangle})^\alpha$ when the i^{th} of p processors is run at speed $s_{\langle i,t\rangle}$.

In addition to considering sublinear nondecreasing speedup curves, [**?**] also considers superlinear and decreasing speedup curves. There are many other related scheduling problems with other objectives, and/or other assumptions about the machine and job instance. Surveys can be found in [**?**, **?**].

2 Proof Sketch

Our goal is to prove that $\forall \text{Alg } \exists \epsilon \ \frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} = \omega(1)$. Believing that LAPS_β (or a triangle version of it) is optimal, we will get all of our intuition from it and from the known lower bounds for it. We know from [?] that LAPS_β is made noncompetitive by giving it extra speed $\epsilon = \frac{\beta}{2}$.

For every algorithm, we need to come up with a constant ϵ such that with this amount of extra speed the competitive ratio R is unbounded. The first complication arises because there is a trade off between these two. See Figure ??. We know that the competitive ratio R can be pushed up to $n^{\Omega(1)}$ but then there is the danger that ϵ will decrease with n. We could simply set ϵ to 10, but then we know that the competitive ratio will be $\mathcal{O}(1)$. To achieve both simultaneously, we fix the competitive ratio R to $\Theta(\ln(n))$, not more and not less, and then minimize the extra speed ϵ needed to achieve this.

Given an arbitrary algorithm Alg, the next challenge is to measure what it does in some way to choose the extra speed ϵ. For LAPS_β, ϵ is set to $\frac{\beta}{2}$ and β is a measure of how much the algorithm concentrates its resources on the later jobs. We will do the same.

Let $\rho_{\langle i,t\rangle}$ denote the resources given at time t to the i^{th} job alive, when the n_t jobs currently alive are sorted by their arrival time. Being speed one, we have that $\sum_{i\in[n_t]} \rho_{\langle i,t\rangle} = 1$. Define $\beta_t = \frac{1}{2}[1 - \frac{1}{n_t}\sum_{i\in[n_t]} i\rho_{\langle i,t\rangle}]$. (This is similar to the potential function used in proving the upper bound for LAPS_β [?].) Define $\beta = \lim_{t\to\infty}\beta_t$. (This measure is cooked so that LAPS_β has measure β.)

The lower bound follows the two that are in the literature. The first will prove that the competitive ratio is $\Omega(\frac{1}{\beta})$ (and $\Omega(\frac{1}{\beta\epsilon})$ if the algorithm favors later jobs) so that when this measure $\beta = \lim_{t\to\infty}\beta_t$ is zero, the competitive ratio is unbounded. The second technique proves that if $\beta = \lim_{t\to\infty}\beta_t$ is a constant, then the competitive ratio is unbounded when $\epsilon = \frac{4}{9}\beta$.

The first technique is referred to as *steady state stream* [?, ?]. We prove that if Alg concentrates its resources on a few jobs as measured by β_j, then there exists at least $\ell = \mathcal{O}(1)$ j jobs that each receive about $\rho_{\langle i,j\rangle} = \frac{1}{\beta_j j}$ fraction of the processors. The input is constructed so that unbeknownst to the nonclairvoyant algorithm these ℓ jobs are sequential. These resources are effectively wasted because the nature of sequential work is that it cant uses these processors. Because of this waste, Alg falls behind on the parallelizable jobs and as such the number of them remaining alive increase. However, to compensate for such mistakes, Alg, has been given ϵ extra resources. It reaches a steady state when the wasted resources equals the extra resources. This occurs when $\frac{\ell}{\beta_j j} = \epsilon$, or $j = \frac{\ell}{\beta_j \epsilon}$. Opt, mean while is completing the parallelizable work as it arrives so only has these ℓ sequential jobs alive. This gives a competitive ratio of $\Omega(\frac{1}{\epsilon\beta_j})$.

The second technique defines a very carefully fine tuned instance *MPT* which was introduced in [?, ?, ?, ?] as a lower bound instance for Equi, $\text{Equi}_{2+\epsilon}$, and LAPS_β. As done in the *steady state stream*, Alg initially falls behind on the stream of parallel jobs because it wastes resources on sequential jobs. However, on this instance the number of alive jobs n_t briefly peeks way above the steady state number because none of the parallelizable jobs complete. This is possible even though the total parallelizable work in the system is decreasing rapidly, because the work remaining per job decreases even faster. This is achieved by having the next jobs to arrive have work approximately equal to the work remaining in the previous jobs. If $\beta = \lim_{t\to\infty}\beta_t$ is a constant, then Alg cannot concentrate too many resources on this new job and hence it falls behind too.

This lower bound also has the same basic circular argument that every lower bounds does. You need to know what the input is before you can know what the algorithm does and you need to know what the algorithm does before you can know on which input it performs poorly. The standard way of breaking this cycle is proceed step by step revealing the minimum amount of information about the input to fix what the algorithm does in the next step. In this paper, we break this classic cycle instead by using Brouwer's fixed point theorem. We tell the non-clairvoyant algorithm the order $J_1, J_2, \ldots$ that the jobs will arrive and that none of these jobs complete. This serves the dual purpose of keeping its flow time high and revealing to it most of the information that it learns as it proceeds. The only things remaining that it learns are the gaps t_i between the times at which the jobs arrival. Being nonclairvoyant, it suffices to specify these $\vec{t} = \langle t_i \mid i \in [1,n] \rangle$, but not the work in the jobs, to know what the algorithm does. What it does is fully specified by $\vec{\rho} = \left\langle \rho_{\langle i,j \rangle} \mid i \in [1,n], j \in [i,n] \right\rangle$, where $\rho_{\langle i,j \rangle}$ is the amount of resources allocated by algorithm Alg to job J_i when there are currently j parallel jobs alive. This information tells us the total amount of work $w_i = \sum_{j \geq i} \rho_{\langle i,j \rangle} t_j$ completed on the job. To ensure that the job is not completed by Alg, we need to set its work to this amount w_i. However, the other concern is that we must keep the the optimal algorithm Opt's flow time down. We do this by assuring that it is able to complete this stream of jobs as they arrive. This is at all possible even though it has speed $\frac{1}{1+\epsilon}$ while Alg has speed 1 because it does not waste any processing power on sequential jobs. Given this speed, it takes Opt time $(1+\epsilon)w_i$ to complete job J_i. However, the only time it has to work on it is the time between when J_i arrives and when J_{i+1} arrives. Hence, we to set the length of this time interval to be $t'_i = (1+\epsilon)w_i$. This completes the cycle, because we started this process by setting the length of this time interval to being t_i. Let $A(\vec{t}) = \vec{t'}$ denote this mapping from these initially stated times $\vec{t}$ to these desired times $\vec{t'}$. What we need are times $\vec{t}$ for which $A(\vec{t}) = \vec{t}$. This is where we use Brouwer's fixed point theorem. Of course to do this, we will have to make the assumption that the algorithm Alg is such that the function A is continuous and we will have to prove that A is bounded within a finite closed space.

Theorem **??** is restricted in that it requires the algorithm to be *graceful* and *deterministic*. We will now formally define these restrictions and argue that they are minor.

1. The distribution $\left\langle \rho_{\langle i,j \rangle} \mid i \in [n], j \geq i \right\rangle$ must be a continuous function of the times $\langle t_i \mid i \in [n] \rangle$. This is true for all algorithms in the literature. Recall that the order that the jobs arrive is fixed and that they have not completed. The only scheduler we can think of that depends at all on the t_i is Shortest Elapsed Time First (SETF) which gives priority to more recently arriving jobs, until they have been processed as much as older jobs, at which point all jobs are given equal priority. This relationship, however, is continuous.

2. $|\frac{\delta \beta'_j}{\delta j}| \leq \frac{\beta}{10j}$.
It would be nice to let the algorithm have β_j be an arbitrary function of j. However, this causes some complications in the proof. See Figure **??**. Hence, we insist that the β_j cannot change too quickly as a function of the number of jobs j currently alive. This is not an unreasonable requirement, however. Every time the number of jobs j doubles, β_j can change by $\frac{\beta}{10}$. Recall, however, that β_j can never change all that much because it is limited between zero and one. Also we see no real advantage for it changing at all. The algorithm LAPS$_\beta$, for example, keeps it constant.

3. Given we define $\beta = \lim_{t \to \infty} \beta_t$, one might thing that we need another constraint that this limit exists. However, the truth, however, is that because of the previous constraint, this one does not matter. Instead, one can simply choose some large n. Because β_j does not change much between $j \in [\frac{1}{2}n, n]$ we can used $\beta = \beta_n$ as our approximation.

4. Being *nondeterministic* in this context only means that the algorithm knows the amount of extra resources ϵ that it is given. We, on the other hand, need to hide this information.

We will now summarize the above intuition.
Def^n Given a fixed past and j parallel jobs alive, let $\beta_j = \frac{1}{j\sum_{i\leq j}\rho^2_{\langle i,j\rangle}}$.
β_j measures how much Alg concentrates its resources on to individual jobs (likely the later ones). This definition is designed so that LAPS$_\beta$ has $\sum_{i\in[1,j-\beta j]} 0^2 + \sum_{i\in[j-\beta j+1,j]}(\frac{1}{\beta j})^2 = \beta j\cdot(\frac{1}{\beta j})^2 = \frac{1}{\beta j}$ and $\beta_j = \beta$.

Theorem 2. *If* $\lim_{t\to\infty}\beta_t < \beta$*, then there exists an* ϵ *and a set of jobs on which* Alg *has a competitive ratio of* $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} > \frac{1}{\beta\epsilon}$.

Note that if $\beta = \lim_{t\to\infty}\beta_t = 0$, then the competitive ratio is infinity.

Theorem 3. *Consider any* graceful deterministic *non-clairvoyant algorithms* Alg*. Suppose* $\beta = \lim_{t\to\infty}\beta_t$ *is a constant, then* $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} = \omega(1)$*, when* $\epsilon = \frac{4}{9}\beta$.

Theorem 4. *We show that triangle* LAPS$_\beta$ *is optimal. See Figure* **??**. *Rectangle* LAPS$_\beta$ *against* Opt$_{1/(1+\epsilon)}$ *needs* $\beta = 2\epsilon$ *and competitive ratio* $\frac{1}{\epsilon\beta} = \frac{1}{2\epsilon^2}$*, while triangle* LAPS$_\beta$ *needs* $\beta = \frac{9}{4}\epsilon$ *and competitive ratio* $\frac{1}{\epsilon\beta} = \frac{4}{9\epsilon^2}$*, which is a factor of* $\frac{8}{9}$ *better.*

Outline: Section **??** describes the *steady state instance* needed for Theorem **??** giving that competitive ratio is $\Omega(\frac{1}{\beta})$ (and $\Omega(\frac{1}{\beta\epsilon})$ if the algorithm favors later jobs). Section **??** describes the *MPT instance* needed for Theorem **??**, giving that speed $1+\epsilon$. This instance is a special case of the steady state instance. Section **??** proves that ϵ is minimized to $\frac{4}{9}\beta$ when the resource distribution strategy $f_j(x)$ is graceful. Section **??** proves that the resource distribution strategy $f_j(x)$ is graceful when β_j changes gracefully. Section **??** uses the fixed point theorem to choose the values for t_i. Some of the proofs have been moved to Section **??** the appendix.

3 Whimsical LAPS

Edmonds and Pruhs [?] introduced a nonclairvoyant algorithm, which they call LAPS$_\beta$, and showed that it is scalable for scheduling jobs with sublinear nondecreasing speedup curves with the objective of total response time. In this section, we stretch their proof to apply for the slightly wider class of algorithms.

LAPS$_\beta$(Latest Arrival Processor Sharing) Definition: The processors are equally partitioned among the $\lceil\beta n_t\rceil$ jobs with the latest arrival times, where n_t is the number of jobs alive at time t and $\beta\in(0,1]$ is a parameter of the algorithm.

The parameter β gives a tradeoff. As β decreases towards zero, *LAPS* favors more recently released jobs in a manner reminiscent of SETF. As such it does not perform well when there are sequential jobs. Equi, which is identical to *LAPS* with $\beta = 1$, spreads the resources two thinly and as such needs $1+\beta+\epsilon$ extra speed to be competitive. Within these restrictions, we relax the definition of LAPS$_\beta$ as follows.

WLAPS$_{\langle\alpha,\beta\rangle}$(Whimsical LAPS) Definition: Let $\rho_{\langle i,t\rangle}$ denote the resources given at time t to the i^{th} job alive, when the n_t jobs currently alive are sorted by their arrival time. The algorithm can allocate its resources at each point in time however it likes under the following three conditions.

1. Speed one: $\sum_{i\in[n_t]}\rho_{\langle i,t\rangle} = 1$.

2. Do not focus too much: $\forall i \in [n_t], \rho_{\langle i,t\rangle} \leq \frac{1}{\alpha i}$.

   A efficient nonclairvoyant algorithm should not give too many resources to any single job because it may be sequential. (At least for the analysis, it is also problematic if the algorithm gets too far ahead of the optimal algorithm on any individual job.)

3. Do not focus too little: $\sum_{i\in[n_t]} i\rho_{\langle i,t\rangle} \geq (1-\frac{\beta}{2})n_t$ (or $\beta_t \overset{\text{def}}{=} 2[1-\frac{1}{n_t}\sum_{i\in[n_t]} i\rho_{\langle i,t\rangle}] \leq \beta$).

   The algorithm should not spread its resources too thinly or else the more recently arriving jobs do not get enough. If a newly arrived job has a small amount of work, then it needs to be completed relatively quickly.

LAPS$_\beta$ has parameters $\alpha = \beta$ and β. The first because $\rho_{\langle i,t\rangle}$ is either zero or $\frac{1}{\beta n_t} \leq \frac{1}{\alpha i}$. The second because $\sum_{i\in[n_t]} i\rho_{\langle i,t\rangle} = \sum_{i\in[(1-\beta)n_t,n_t]} i\frac{1}{\beta n_t} = \frac{1}{\beta n_t}[\frac{i^2}{2}]_{i=(1-\beta)n_t}^{n_t} = \frac{1}{2\beta n_t}[(n_t)^2 - ((1-\beta)n_t)^2] = \frac{n_t}{2\beta}[1-[1-2\beta+\beta^2]] = \frac{n_t}{2\beta}[2\beta-\beta^2] = n_t[1-\frac{\beta}{2}]$. A quick modification of result in [?] gives the following.

Theorem 5. *Whimsical LAPS with parameters α and β and $s = (1+\beta+\epsilon)$ times as many processors as the optimal is a $\left(\frac{4s}{\alpha\epsilon}\right)$-competitive algorithm for scheduling processes with sublinear nondecreasing speedup curves for the objective of average response time.*

Proof. Their proof uses an amortized local competitiveness argument with a simple potential function, Φ_t. The decrease in Φ_t due to WLAPS$_{\langle\alpha,\beta\rangle}$'s processing is defined to be $-\frac{d\Phi_t}{dt} = \sum_{i=[1,n_t]-L_t} i\rho_{\langle i,t\rangle}$ where L_t denotes the set of jobs on which WLAPS$_{\langle\alpha,\beta\rangle}$ is executing a sequential phase at this time or WLAPS$_{\langle\alpha,\beta\rangle}$ is ahead of Opt on this job at this time. By the parameters of the algorithm this decrease is at least $\sum_i i\rho_{\langle i,t\rangle} - |L_t|\cdot\max_i i\rho_{\langle i,t\rangle} \geq (1-\frac{\beta}{2})n_t - \frac{|L_t|}{\alpha}$. The rest of the proof follows the same. □

4 The Steady State Technique

Theorem ??: *If $\lim_{t\to\infty}\beta_t \leq \beta$, then even if* Alg *is given s times as much resources for some large s, there still exists a set of jobs on which* Alg *has a competitive ratio of $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/s}(J))} \geq \Omega(\frac{1}{\beta})$. If the algorithm favors the most recently arriving jobs, then this lower bound can be improved to $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} \geq \Omega(\frac{1}{\epsilon\beta})$.*

This result is tight with the for large s, but not for $s \approx 1+2\beta$.

** When $s = \gamma+1$, then all jobs get complete in place and competitive ratio is 1. β is $2(1-\frac{1}{s}) \approx 2$. Hence the statement is not false, but not meaningful. What if we had Q copies of this same job set so that β has more meaning. Actually, if alg had small β then it would give all resources to most recent sequential job and get nowhere.

If the algorithm favors the most recently arriving jobs, which is the case for every algorithm considered in the literature.

Proof. The concept of a *steady state stream* was introduced in [?]. Not only does it provide a lower bound, it also provides intuition as to why LAPS$_\beta$ manages to work at all. It is perhaps hard to believe that a nonclairvoyant scheduler, even with more processors, can perform well here. The scheduler does not know which of the jobs is parallelizable. Hence it wastes most of the processors on the sequential jobs, and falls further and further behind on the parallelizable jobs. Alg, however, is able to automatically "self adjust" the number of processors wasted on the sequential jobs so that it eventually reaches a *steady state*. The definition of steady state is that Alg is completing parallelizable work at the rate that it arrives, where the rate that it arrives is $\frac{1}{1+\epsilon}$, because this is the rate at which Opt is able to complete it as it arrives. Let $waste_t$ denote the total resources wasted on sequential jobs at time t. The Alg has $\epsilon' = 1 - \frac{1}{1+\epsilon} = 1 - \frac{1}{s}$ extra resources. Hence, Alg reaches a steady state when $waste_t = \epsilon'$.

For this lower bound, it is sufficient to observe that if Alg is not in such a steady state, then the amount parallelizable work that has arrived and not completed continues to increase. Because the initial work w_i in each parallelizable job does not increase with their arrival time, the number n_t of uncompleted jobs continues to increase. Once in the steady state, it can stay there for a long enough time so that these costs dominate the cost of reaching the steady state.

The input is defined by the parameters $\vec{t} \in R^n$, $\epsilon > 0$, $\gamma : \mathcal{Z} \to \mathcal{Z}$, and $\ell \in \mathcal{O}(1)$. It contains a stream of n parallelizable jobs. For each $i \in [n]$, job J_i is released at time $r_i = \sum_{j=1}^{i-1} t_j$ with parallelizable work $\frac{1}{1+\epsilon}t_i$. For each $i \in [n]$, ℓ sequential jobs are released at the same time r_i that the parallelizable job J_i is released. The amount of work in each of these is designed to be such that when there are n_t parallelizable jobs alive, $\ell\gamma(n_t)$ sequential jobs are still alive. If, for example, none of the parallelizable jobs have completed, then there j parallelizable jobs still alive during the period $[r_j, r_{j+1}]$ of length t_j and hence for all $i \in [j - \gamma(j), j]$, the ℓ sequential jobs that arrived at the same time as J_i are still alive. We assume that $\gamma(j) << j$ and hence, the inverse of $i = j - \gamma(j)$ is approximately $j = i + \gamma(i)$ and the sequential jobs released at time r_i will have work $\sum_{i' \in [i,j]} t_{i'}$ so that they complete at time r_{j+1}. Suppose, on the other hand, that some of the parallelizable jobs complete to the point that there are only n_t of them remaining at time t. Let P_t be the set containing the n_t indexes of these jobs. Let $S_t \subseteq P_t$ be the $\gamma(n_t)$ of these that arriving the most recently. For each $i \in S_t$, the ℓ sequential jobs that arrived at the same time as J_i are still alive. Being nonclairvoyant, the algorithm knows nothing about the work in a job until the moment it completes. Hence, the adversary can easily dynamically adjust work in the sequential jobs so that this is the case.

For each time t and each $i \in [n_t]$, let $\rho_{\langle i,t\rangle}$ denote the resources allocated to the i^{th} parallelizable job that is alive at time t, where there jobs are sorted by arrival time. We now want to prove that Alg in the worst case allocates this same amount $\rho_{\langle i,t\rangle}$ to each of the ℓ sequential jobs with the same release time assuming that they are still alive. Being nonclairvoyant, Alg is unable to distinguish between these $1 + \ell$ jobs. Hence, the adversary can switch the one receiving the least during the time that the sequential jobs are alive to be the parallel one. It follows that the total resources wasted on sequential jobs is at least $waste_t = \sum_{i \in [n_t - \gamma(n_t), n_t]} \ell\rho_{\langle i,t\rangle}$.

The algorithm worth noting we call the *delay algorithm*. It achieves competitive ratio $\Omega(\frac{1}{\beta})$ on this set of jobs. It notes that if no processors are given to the jobs than the sequential jobs will complete on their own. Once they are done, the algorithm knows that the remaining jobs are parallelizable. In this way it wastes no resources on the sequential jobs and as such is immediately in a steady state.

At steady state, this waste is equal to the amount of extra resources ϵ'. This gives that $\sum_{i \in [n_t - \gamma(n_t), n_t]} \rho_{\langle i,t\rangle} \le \frac{\epsilon'}{\ell}$. From this we get that $\sum_{i \le [n_t]} i \cdot \rho_{\langle i,t\rangle} \le \sum_{i \in [1, n_t - \gamma(n_t) - 1]} [n_t - \gamma(n_t)] \cdot \rho_{\langle i,t\rangle} + \sum_{i \in [n_t - \gamma(n_t), n_t]} [n_t] \cdot \rho_{\langle i,t\rangle} = [n_t - \gamma(n_t)] \cdot [1 - \frac{\epsilon'}{\ell}] + [n_t] \cdot [\frac{\epsilon'}{\ell}] = n_t - \gamma(n_t) \cdot [1 - \frac{\epsilon'}{\ell}]$. This is more or less tight for the *delay algorithm*. Recall that $2\beta_t = 1 - \frac{1}{n_t} \sum_{i \le [n_t]} i\rho_{\langle i,t\rangle}$. This gives $2\beta_t \ge 1 - \frac{1}{n_t}\left[n_t - \gamma(n_t) \cdot [1 - \frac{\epsilon'}{\ell}]\right] = \frac{\gamma(n_t)}{n_t} \cdot [1 - \frac{\epsilon'}{\ell}] \ge \frac{\gamma(n_t)}{2n_t}$.

We now bound the competitive ratio. At time t, there are n_t parallelizable jobs $\ell\gamma(n_t)$ and sequential jobs alive under Alg. Under Opt, there are the same $\ell\gamma(n_t)$ sequential jobs alive but only one parallelizable job. If we continue being in this steady state for a long time then these costs dominate. This gives that the competitive ratio is $\frac{F(\mathrm{Alg}_1(J))}{F(\mathrm{Opt}_{1/s}(J))} = \frac{n_t + \ell\gamma(n_t)}{1 + \ell\gamma(n_t)} \ge \frac{n_t}{\ell\gamma(n_t)} \ge \Omega(\frac{1}{\beta})$ as required for the first result.

Now suppose that the algorithm favors the most recently arriving jobs, i.e. for all i and t, $\rho_{\langle i,t\rangle} \leq \rho_{\langle i+1,t\rangle}$. The delay algorithm, for example, is not allowed. As we did above, let us maximize $\sum_{i\leq[n_t]} i \cdot \rho_{\langle i,t\rangle}$ subject to $\sum_{i\in[n_t-\gamma(n_t),n_t]} \rho_{\langle i,t\rangle} \leq \frac{\epsilon'}{\ell}$.

We maximize $\sum_{i\leq[n_t]} i \cdot \rho_{\langle i,t\rangle}$ by moving as must resources as possible forward to jobs with the largest coefficients, i.e. to more recently arriving jobs. The constraint $\sum_{i\in[n_t-\gamma(n_t),n_t]} \rho_{\langle i,t\rangle} \leq \frac{\epsilon'}{\ell}$, however, imposes a barrier at the job index $i = n_t - \gamma(n_t) - 1$ past which resources can't be moved. Hence, let ρ' denote this optimal allocation $\rho_{\langle n_t-\gamma(n_t)-1,t\rangle}$ to this job, i.e. to the first parallelizable jobs whose corresponding sequential jobs have completed. Because the algorithm favors the most recently arriving jobs, we know that $\rho_{\langle i,t\rangle} \leq \rho'$ for all $i \in [1, n_t - \gamma(n_t) - 1]$. The resources to these jobs will be moved forward so that for all $i \in [1, i_{min} - 1]$ $\rho_{\langle i,t\rangle} = 0$ and for all $i \in [i_{min}, n_t - \gamma(n_t) - 1]$ $\rho_{\langle i,t\rangle} = \rho'$. Similarly, $\rho_{\langle i,t\rangle} \geq \rho'$ for all $i \in [n_t - \gamma(n_t), n_t]$. The resources to these jobs will be moved forward so that for all $i \in [n_t - \gamma(n_t), n_t - 1]$ $\rho_{\langle i,t\rangle} = \rho'$ increasing the most recent job's allocation to $\rho_{\langle n_t,t\rangle} \geq \rho'$. It takes a bit more calculus, but in fact $\sum_{i\leq[n_t]} i \cdot \rho_{\langle i,t\rangle}$ by setting $\rho_{\langle n_t,t\rangle} = \rho'$ as well. This gives that the optimal algorithm is LAPS. More over, the statement of the theorem defines β_t for this algorithm, giving $\rho_{\langle i,t\rangle} \leq \frac{1}{\beta_t n_t}$ for all $i \in [(1-\beta_t)n_t, n_t]$. The constraint $\frac{\epsilon'}{\ell} \geq \sum_{i\in[n_t-\gamma(n_t),n_t]} \rho_{\langle i,t\rangle} = \gamma(n_t)\frac{1}{\beta_t n_t}$ can be used to bound the competitive ratio $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/s}(J))} \geq \frac{n_t}{\ell\gamma(n_t)} \geq \Omega(\frac{1}{\epsilon'\beta})$ as required for the second result. □

5 Stuff

Intuition: As explained in the introduction, we modify the *MPT* instance [?, ?, ?, ?] to lower bound the performance of any β-algorithm Alg. As with Theorem **??**, it contains a stream of $n - \ell$ parallelizable jobs. For each $i \in [\ell+1, n]$, job J_i is released at time $r_i = \sum_{j=\ell+1}^{i-1} t_j$ with parallelizable work $\frac{1}{1+\epsilon}t_i$, where the values t_i are carefully defined later. As before, it could have some sequential jobs in order to distract Alg away from the stream parallelizable work. In Theorem **??**, the algorithm set β small, meaning that it was focusing on the newly arriving jobs. This is why the proof of this theorem had the sequential jobs be the most recently arrived jobs. Now, however, the algorithm set β large, meaning that it is giving some (though likely a small amount) of its resources to the old job. This is why this proof can have the sequential jobs be the oldest jobs. Unlike the question which jobs are newest jobs, which are the oldest does not change as more jobs arrive. Hence, we can have these sequential jobs be the first ℓ to arrive. Because there are only ℓ of them for some small constant ℓ, it wont hurt the adversary by more than a factor of ℓ to make these constant.

6 The MPT Instance

Theorem ??. *Consider any* graceful deterministic *non-clairvoyant algorithms* Alg. *Suppose* $\beta = \lim_{t\to\infty} \beta_t$ *is a constant, then* $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} = \omega(1)$*, when* $\epsilon = \frac{4}{9}\beta$*. This is true even if all the jobs are parallelizable.*

Proof. As explained in the introduction, we modify the *MPT* instance [?, ?, ?, ?] to lower bound the performance of any β-algorithm Alg. As with Theorem **??**, it contains a stream of $n-\ell$ parallelizable jobs. For each $i \in [\ell+1, n]$, job J_i is released at time $r_i = \sum_{j=\ell+1}^{i-1} t_j$ with parallelizable work $\frac{1}{1+\epsilon}t_i$, where the values t_i are carefully defined later. The input also has ℓ extra jobs identical to the first stream job $J_{\ell+1}$, i.e. arrives at time zero with $\frac{1}{1+\epsilon}t_{\ell+1}$ parallelizable work.

$\text{Opt}_{1/(1+\epsilon)}$ ignores the extra ℓ jobs until the end. Using all of its $1/(1+\epsilon)$ processors, it can complete the parallelizable stream in place. Opt flow time during the stream is $\sum_{i=1}^{n}(\ell+1)t_i$, because it always has only $\ell+1$ jobs alive. In the end, it must complete the ℓ extra jobs. It can complete each of these in t_1 time for an additional flow of $\frac{\ell^2}{2}t_1$. Hence the total is at most $\ell^2\sum_{i=1}^{n}t_i$.

In contrast, Alg does not do as well. By the statement of the theorem, it sets its parameter β large, meaning that it gives some (though likely a small amount) of its resources to older jobs. When the number of jobs alive is small relative to ℓ, the algorithm must be giving some of its resources to the ℓ extra jobs. We consider this to be a waste because it has no hope of completing these jobs before the stream is over. Because of this waste, the algorithm falls so hopelessly behind on the earliest arriving stream jobs that it can never complete these either. This problem snow balls. As more and more stream jobs arrive, the large β means that the algorithm allocates some of its resources to the older already hopeless stream jobs. Hence, it falls hopelessly behind on these newly arrived stream jobs as well. This snowballing is all extremely finely orchestrated so that Alg manages to complete none of the jobs during the stream. This results in a flow time of at least $[\sum_{i=1}^{n}it_i]$.

Lemma **??** then carefully constructs the values t_i with the following conditions.

0. Distribution of Resources: When the jobs arrive according to $\langle t_i \mid i\in[\ell+1,n]\rangle$, Alg distributes its resources according to $\left\langle \rho_{\langle i,j\rangle}\geq 0 \mid j\in[\ell+1,n], i\in[1,j]\right\rangle$. Here $\rho_{\langle i,j\rangle}$ denotes the average number of processors allocated to job J_i during the time period $[r_j, r_j+t_j]$.

1. Don't Complete: $\forall i\in[\ell+1,n],\ w_i \overset{\text{def}}{=} \sum_{j\geq i}\rho_{\langle i,j\rangle}t_j \leq \frac{1}{1+\epsilon}t_i$.
   All of these inequality will be tight except for $i=n$.
   Here w_i is the amount of work that Alg has done on job J_i by time r_{n+1} and $\frac{1}{1+\epsilon}t_i$ is the amount of work in the job. This ensures that as promised, the job does not complete under Alg during the stream. This is proved in Lemma **??**.

   The first $\ell+1$ jobs are identical. The adversary let $J_{\ell+1}$ denote the one that is allocated the most resources by Alg. Given $J_{\ell+1}$ does not complete, neither do the ℓ jobs.

2. Speed one: $\forall j,\ \sum_{i\in[j]}\rho_{\langle i,j\rangle}=1$.

3. Def^n of β: $\forall j,\ \sum_{i\in[j]}i\rho_{\langle i,j\rangle}=(1-\frac{\beta_j}{2})j$.
   This is by definition of β_j.

4. Total Time: $\sum_i t_i = 1$
   There is no harm is scaling the jobs so that this is true.

5. $F(\vec{t}) \overset{\text{def}}{=}= \sum_i it_i = \ln(n)$
   This assures that the competitive ratio is $\frac{F(\text{Alg}_1(J))}{F(\text{Opt}_{1/(1+\epsilon)}(J))} \geq \frac{\sum_i it_i}{\ell^2\sum_i t_i} = \Omega(\ln n)$.

6. $Loss(\vec{t}) \overset{\text{def}}{=} \sum_{i\in[1,\ell]}\sum_{j\in[\ell+1,n]}$.

□

Lemma 6. *Conditions[1&5] requires that* $\epsilon \leq \frac{\beta}{2}$

Lets set $\epsilon = \frac{\beta}{4}$ in order to give us some slack.

Proof. Multiplying condition[1] by i and summing gives that
$\sum_i iw_i = \sum_i i \sum_{j \geq i} \rho_{\langle i,j \rangle} t_j \leq \sum_i i \frac{1}{1+\epsilon} t_i$
$\sum_j \left[\sum_{i \leq j} i\rho_{\langle i,j \rangle}\right] t_j \leq \sum_i i \frac{1}{1+\epsilon} t_i$
Condition[5] then gives
$\sum_j \left[j[1 - \frac{\beta}{2}\right] t_j \leq (1-\epsilon) \sum_i it_i$
$1 - \frac{\beta}{2} \leq 1 - \epsilon$
$\epsilon \leq \frac{\beta}{2}$. □

7 The Resource Distribution Strategy $f_j(x)$

Lemma 7. *Suppose that* $|\frac{\delta \beta'_j}{\delta j}| \leq \frac{\beta}{10j}$ *and that* $\beta = \lim_{n \to \infty} \beta_j$ *is a constant. Subject to the constraints[1,6] stated in the proof of Theorem* **??**, ϵ *is minimized to* $\frac{4}{9}\beta$. *In fact, (triangle)* LAPS$_{\beta_j}$ *is optimal.*

Def^n The *resource distribution strategy* f_j: For each j and $x \in [0,1]$, let $f_j(x) = j\rho_{\langle xj,j \rangle}$. For example, LAPS$_\beta$ has $f(x) = 0$ for $x \in [0, 1-\beta]$ and $f(x) = \frac{1}{\beta}$ for $x \in [1-\beta, 1]$.
The change of variables with $i = xj$ and $\delta i = j\delta x$ gives the following conversions.

Lemma 8.

1. $\int_{x \in [0,1]} f_j(x)\delta x = \int_{i \in [0,j]} \left[j\rho_{\langle xj,j \rangle}\right] \left[\frac{\delta i}{j}\right] = \int_{i \in [0,j]} \rho_{\langle xj,j \rangle} \delta i = 1$. *This uses constraint[2].*

2. $\int_{x \in [0,1]} xf_j(x)\delta x = \int_{i \in [0,j]} \left[\frac{i}{j}\right] \left[j\rho_{\langle xj,j \rangle}\right] \left[\frac{\delta i}{j}\right] = \frac{1}{j} \left[\int_{i \in [0,j]} i\rho_{\langle xj,j \rangle} \delta i\right] = \frac{1}{j} \left[j(1 - \frac{\beta_j}{2})\right] = (1 - \frac{\beta_j}{2})$
This uses constraint[2].

We can now prove Lemma **??**.

Proof. An initial concern is that because the fixed point theorem gives us the t_i, we don't know what they are. However, after getting the $\rho_{\langle i,j \rangle}$ and ϵ, the t_i can be uniquely determined using back substitution into the equations $w_i = \sum_{j \geq i} \rho_{\langle i,j \rangle} t_j = \frac{1}{1+\epsilon} t_i$. Computing these t_i would be hard. However, it is sufficient to guess the t_i and check that the required equations hold. Set $t_i = \frac{c}{i^{q_i}}$, where $q_i = 1 + \frac{\beta}{\beta_i}$. Note that $\lim_{n \to \infty} q_j = 2$.

We first check that $\frac{\sum_i it_i}{\sum_i t_i} \geq \ln n$ holds. Note that if β_i was equal to the constant β, then effectively $q_i = 2$. This then gives $\sum_i t_i = 1 + \sum_i \frac{1}{i^2} = \Theta(1)$ and $\sum_i it_i = 1 \cdot 1 + \sum_i i\frac{1}{i^2} = \Theta(\ln n)$. Hence, as needed $\frac{\sum_i it_i}{\sum_i t_i} = \Theta(\ln n)$. Now more generally suppose $\lim_{i \to n} \beta_i = \beta$. Then there is some i' after which β_i is effectively β. Hence as before, $\sum_{i \leq i'} t_i = \Theta(1)$, $\sum_{i > i'} t_i = \sum_{i > i'} \frac{1}{i^2} = \Theta(1)$, $\sum_{i \leq i'} it_i = \Theta(1)$, and $\sum_{i > i'} it_i = \sum_{i > i'} i\frac{1}{i^2} = \Theta(\ln n)$.

Let us now check $w_i = \sum_{j \geq i} \rho_{\langle i,j \rangle} t_j \leq \frac{1}{1+\epsilon} t_i$. This requires a change of variables with $j = \frac{i}{x}$ and $\delta j = -\frac{i}{x^2}\delta x$. Lemma **??** proves that that the algorithm's resource distribution strategy f_j is either independent of j or changes slowly enough that f_j can be approximated by f_i. Lemma **??** (in the appendix) bounds $\int_{x=0}^{1} x^{q-1} f(x)\delta x$.

$$\begin{aligned} w_i &= \sum_{j \geq i} \rho_{\langle i,j \rangle} t_j = \int_{j=i}^{\infty} \frac{f_j(\frac{i}{j})}{j} \cdot \left(\tfrac{1}{j}\right)^{q_i} \delta j = \int_{j=i}^{\infty} f_j\left(\tfrac{i}{j}\right) \cdot \left(\tfrac{1}{j}\right)^{q_i+1} \delta j = \int_{x=1}^{0} f_i(x) \cdot \left(\frac{x}{i}\right)^{q_i+1} \left[-\frac{i}{x^2}\delta x\right] \\ &= \left[\int_{x=0}^{1} x^{q_i-1} f_i(x)\delta x\right] t_i \leq \left[1 - \frac{4}{9}(q_i - 1)\beta_i\right] t_i = \left[1 - \frac{4}{9}\beta\right] t_i \approx \frac{1}{1+\epsilon} t_i \end{aligned}$$

$\square$

8 Lagrange Multipliers

The lower bounds apply as long as $\epsilon \approx 1 - \frac{1}{1+\epsilon} < \frac{\beta}{2}$. We express one over the gap between these amounts with $a = 1/(\frac{1}{1+\epsilon} - (1 - \frac{\beta}{2}))$.

Lemma 9. *Suppose $\beta_j = \beta$ is a constant independent of j and constraint[1] is tight, i.e. each job just finishes at time $r_n + 1$. Subject to the constraints[1,4], the amount $F(\vec{t}) + aLoss$ surprisingly is a constant independent of how the adversary sets the $\vec{t}$ algorithm sets the $\vec{t}$. Specifically, its value agrees with that in given in Lemma* **??**. *More over, this amount increases when a constraint $C1_j$ is not tight and decreases when a parameter β_j increase.*

Recall that $F(\vec{t}) = \omega(1)$, $Loss \in [0,1]$, and a is a constant. Hence, $F(\vec{t}) + aLoss \approx F(\vec{t})$.

BUG!!! If you give these equations to Maple, they say that $F(\vec{t}) + aLoss = 0$. Looking at EQUI, we see C3 should actually have $(j+1)(1 - \frac{\beta}{2})$. This improves things slightly. giving $F(\vec{t}) + aLoss = \frac{1-\beta}{a} = \frac{1-\beta}{\frac{\beta}{2}-\epsilon}$. But how can this be the value for every setting of $\vec{t}$ and $\vec{\rho}$ when we have setting which give $F(\vec{t}) = \ln(n)$. Plugging in EQUI with $\rho_{\langle i,j\rangle} = \frac{1}{j}$, maple says there are no values of $\vec{t}$ satisfying the equations. I guess $\sum_{j\in[i,n]} \rho_{\langle i,j\rangle} t_j = \frac{t_i}{1+\epsilon}$ does not hold exactly.

Proof. The Lagrange multipliers method says to take as follows a linear combination of the goal and each of the constraints.

The Kuhn-Tucker Conditions state (http://www.economics.utoronto??)
Max $f(x)$ subject to $g_j(x) \le c_j$ for $j = 1, \dots, m$
are $L_i'(n) = 0$ for $i = 1, \dots, n$
$\lambda_j \ge 0$, $g_j(x) \le c_j$, and $\lambda_j(g_j(x) - c_j) = 0$, for $j = 1, \dots, m$
where $L(x) = f(x) + \sum_{j=1}^{m} \lambda_j(g_j(x) - c_j)$.
The key is that constraint g_j can be relaxed if $\lambda_j \ge 0$.

$$\begin{aligned} G &= \left[F(\vec{t}) + aLoss\right] + \sum_{i\in[\ell+1,n]} \gamma_i \tfrac{(C1_i)}{1+\epsilon} + \sum_{j\in[\ell+1,n]} \lambda_j(C2_j) - \sum_{j\in[\ell+1,n]} \alpha_j(C3_j) + W(C4) \\ &= \sum_{j\in[\ell+1,n]} jt_j \;+\; a \sum_{i\in[1,\ell]} \sum_{j\in[\ell+1,n]} i\rho_{\langle i,j\rangle} t_j \\ &\quad + \sum_{i\in[\ell+1,n]} \gamma_i \left[\sum_{j\in[i,n]} \rho_{\langle i,j\rangle} t_j - \tfrac{t_i}{1+\epsilon}\right] + \sum_{j\in[\ell+1,n]} \lambda_j \left[\sum_{i\in[1,j]} \rho_{\langle i,j\rangle} - 1\right] \\ &\quad - \sum_{j\in[\ell+1,n]} \alpha_j \left[\sum_{i\in[1,j]} i\rho_{\langle i,j\rangle} - j\left(1 - \tfrac{\beta}{2}\right)\right] \;+\; W \left[\sum_{j\in[\ell+1,n]} t_j - 1\right] \end{aligned}$$

The Lagrange multipliers method goes on to say that the optimal occurs when the derivative of G with respect to each variable is zero. This gives two new sets of conditions on the optimal values.

6. $\forall i \in [1,\ell],\ j \in [\ell+1,n]$ and $\langle i,j\rangle = \langle n,n\rangle$,

$$\frac{\delta G}{\delta \rho_{\langle i,j\rangle}} = ait_j + \lambda_j - \alpha_j i = 0 \quad \text{or} \quad i\alpha_j = ait_j + \lambda_j$$

Using this expression, we can fix many of our parameters. This equation with $i = 2$ minus that with $i = 1$ gives $(2-1)\alpha_j = a(2-1)t_j$, which gives for $j \in [\ell+1, n]$, $\alpha_j = at_j$. Plugging this back in gives $\lambda_j = 0$.

6'. $\forall i \in [\ell+1, n],\ j \in [i, n]$,

$$\frac{\delta G}{\delta \rho_{\langle i,j\rangle}} = \gamma_i t_j + \lambda_j - \alpha_j i = \gamma_i t_j + (0) - (at_j)i = 0 \ \text{ or } \ \gamma_i = ai$$

7 $\forall j \in [\ell+1, n]$,

$$\begin{aligned}\frac{\delta G}{\delta t_j} &= j \ + \sum_{i\in[1,\ell]} ai\rho_{\langle i,j\rangle} \ + \sum_{i\in[\ell+1,j]} \gamma_i \rho_{\langle i,j\rangle} \ - \ \gamma_j \tfrac{1}{1+\epsilon} \ + \ W \\ &= j \ + \sum_{i\in[1,j]} ai\rho_{\langle i,j\rangle} \ - \ aj\tfrac{1}{1+\epsilon} \ + \ W \\ &= j \ + \ (aj(1-\tfrac{\beta}{2})) \ - \ aj\tfrac{1}{1+\epsilon} \ + \ W \ = \ 0 + W \ = \ 0\end{aligned}$$

The surprising thing is that we were able to show that this is a minimum independent of how the algorithm sets the $\rho_{\langle i,j\rangle}$. $\square$

9 Original Lagrange Multipliers

This was the original version when the measure was $\sum_{i\in Z|_j} \rho^2_{\langle i,j\rangle} = \frac{1}{\beta}$. It manages to tell you what $\vec{\rho}$ is supposed to be.

Lemma 10. *Suppose* $|\frac{\delta \beta'_j}{\delta j}| \leq \frac{\beta}{10j}$. *Subject to the constraints[1,6], the algorithm's resource distribution strategy f_j changes slowly enough that f_j can be approximated by f_i.*

Proof. Let us again minimize ϵ subject to the constraints[1,6]. Let us begin by using constraint[6] that $\rho_{\langle i,j\rangle} \geq 0$ to define the set $Z = \{\langle i,j\rangle \mid \rho_{\langle i,j\rangle} > 0\}$. (If Alg is (triangle) LAPS$_\beta$ as we claim, then Z will consist of $i \in [(1-\beta)j, j]$.) See Figure **??** (in the appendix).

The Lagrange multipliers method says to take as follows a linear combination of the goal and each of the constraints.
$G = \epsilon \ + \ \sum_i \gamma_i \left[\sum_{j\in Z|_i} \rho_{\langle i,j\rangle} t_j - (1-\epsilon)t_i\right] \ - \ \sum_j \lambda_j \left[\sum_{i\in Z|_j} \rho_{\langle i,j\rangle}] + [\max_i \rho_{\langle i,j\rangle}]\ell - 1\right]$
$- \ \sum_j \frac{1}{2}\alpha_j \left[\sum_{i\in Z|_j} \rho^2_{\langle i,j\rangle} - \frac{1}{\beta}\right] \ - \ W\left[\sum_i it_i - \ln(n)\right] \ - \ W'\left[\sum_i t_i - 1\right]$.
The Lagrange multipliers method goes on to say that the optimal occurs when the derivative of G with respect to each variable is zero. This gives two new conditions on the optimal values.

7. $\forall \langle i,j\rangle \in Z$, $\frac{\delta G}{\delta \rho_{\langle i,j\rangle}} = t_j\gamma_i - \lambda_j - \rho_{\langle i,j\rangle}\alpha_j = 0$, or $\rho_{\langle i,j\rangle} = \max\left(\frac{t_j\gamma_i - \lambda_j}{\alpha_j}, 0\right)$.

8. $\forall j$, $\frac{\delta G}{\delta t_j} = \sum_{i\leq j} \rho_{\langle i,j\rangle}\gamma_i - jW - W' = 0$, or $\sum_{i\leq j} \rho_{\langle i,j\rangle}\gamma_i = jW + W'$.

Lemma **??** (in the appendix) proves that these new constraints[7,8] are satisfied by the same triangle LAPS$_\beta$ algorithm shown in Lemma **??** to satisfy constraints[1,6]. We, however, are more concerned with showing that f_j changes slowly enough that f_j can be approximated by f_i. Suppose the parameters $\langle \gamma_i \mid i \in [n]\rangle$ were determined. The key thing about these values γ_i is that they do not depend on j. Hence, constraint[7] states that the distribution $\rho_{\langle 1,j\rangle}, \ldots, \rho_{\langle j,j\rangle}$ and hence $f_j(x)$

depend on j via only two parameters $\frac{t_j}{\alpha_j}$ and $\frac{\lambda_j}{\alpha_j}$. Condition[2], stating that $\sum_i \rho_{\langle i,j\rangle} = 1$, can be used to determine the first of these parameters (as a function of the second) and Condition[3], stating that $\sum_{i\leq j} \rho^2_{\langle i,j\rangle} = \frac{1}{j\beta_j}$, can be used to completely determine $f_j(x)$ as a function β_j (and the parameters $\langle \gamma_i \mid i \in [n]\rangle$). By the condition of the lemma, $|\frac{\delta\beta'_j}{\delta j}| \leq \frac{\beta}{10j}$. Hence, as a function of j, $f_j(x)$ cannot change quickly. When the algorithm is triangle LAPS$_{\beta_j}$, Lemma **??** (in the appendix) proves that this suffices to show that $f_j(x)$ changes sufficiently slowly to be approximated by $f_i(x)$. □

10 Zeros of Lagrange Multipliers

Def^n A *Manhattan path of non-zero* $\rho_{\langle i,j\rangle}$ consists of a path of tuples $\langle i_k, j_k\rangle$ such that

1. $\langle i_1, j_1\rangle = \langle 1, \ell+1\rangle$ and $\langle i_{2n-l}, j_{2n-l}\rangle = \langle n-1, n\rangle$,
2. $\forall k \in [1, 2n-\ell-1]$, either ($i_{k+1} = i_k$ and $j_{k+1} = j_k+1$) or ($i_{k+1} = i_k+1$ and $j_{k+1} = j_k$), and
3. $\forall k \in [1, 2n-\ell]$, $\rho_{\langle i_k,j_k\rangle} \neq 0$ and $\rho_{\langle i_{k+1},j_k\rangle} \neq 0$.

Lemma 11. *Suppose:*

1. *There is a* Manhattan path of non-zero $\rho_{\langle i,j\rangle}$.
2. $\forall i \in [1,\ell],\ j \in [\ell+1, n]$ and $\langle i,j\rangle = \langle n,n\rangle$, if $\rho_{\langle i,j\rangle} \neq 0$, then $i\frac{t_j}{a} = i\alpha_j + \lambda_j$.
3. $\forall i \in [\ell+1, n-1],\ j \in [i,n]$, if $\rho_{\langle i,j\rangle} \neq 0$, then $\gamma_i t_j = i\alpha_j + \lambda_j$

then

1. $\forall i \in [1,n]$, $\gamma_i = \frac{i}{a'} + b$.
2. $j \in [\ell+1, n]$, $\alpha_j = \frac{t_j}{a}$ *and* $\lambda_j = 0$.

Proof. The proof is by induction on k that the statement is true for $i \in [1, i_k]$ and $j \in [\ell+1, j_k]$. The base case is $k=1$ and $\langle i_1, j_1\rangle = \langle 1, \ell+1\rangle$. We have $\rho_{\langle 1,\ell+1\rangle} \neq 0$ and $\rho_{\langle 2,\ell+1\rangle} \neq 0$, and hence that $1\frac{t_{\ell+1}}{a} = 1\alpha_{\ell+1} + \lambda_{\ell+1}$ and $2\frac{t_{\ell+1}}{a} = 2\alpha_{\ell+1} + \lambda_{\ell+1}$. subtracting and plugging back in give $\alpha_{\ell+1} = \frac{t_{\ell+1}}{a}$ and $\lambda_{\ell+1} = 0$.

Using this expression, we can fix many of our parameters. This equation with $i=2$ minus that with $i=1$ gives $(2-1)\frac{t_j}{a} = (2-1)\alpha_j$, which gives for $j \in [\ell+1, n]$, $\alpha_j = \frac{t_j}{a}$. Plugging this back in gives $\lambda_j = 0$.

Using this expression, we can fix many of our parameters. This equation with $i=2$ minus that with $i=1$ gives $(2-1)\frac{t_j}{a} = (2-1)\alpha_j$, which gives for $j \in [\ell+1, n]$, $\alpha_j = \frac{t_j}{a}$. Plugging this back in gives $\lambda_j = 0$.

Using this expression, we can fix many of our parameters. This equation with $i=\ell+2$ minus that with $i=\ell+1$ gives $(\gamma_{\ell+2} - (\gamma_{\ell+1})t_j = \alpha_j$. This gives that for $j \in [\ell+2, n]$, $\alpha_j = a't_j$ for some constant a'. This equation with i and $j=n$ minus that with $i=\ell+1$ gives $(\gamma_i - (\gamma_{\ell+1})t_n = \alpha_n$. This gives that for $i \in [\ell+1, n]$, $\gamma_i = \frac{i}{a'} + b$ for some constant b. Plugging these in for $i = \ell+1$ gives $(\frac{i}{a'} + b)(a'\alpha_j) = i\alpha_j + \lambda_j$. This gives that for $j \in [\ell+2, n]$, $\lambda_j = a'b\alpha_j = bt_j$. For $j \in \ell+1$, we do not have $\alpha_j = a't_j$.

□

11 The Fixed Point Theorem

Lemma 12. *For every nonclairvoyant algorithm* Alg *such that the total work w_i done on the i^{th} job is a continuous function of the times $\langle t_i \mid i \in [n]\rangle$, there are values t_i that satisfy constraints[0,6].*

Proof. Fix parameters n and $\ell = \omega(1)$.
Def^n: Let $B = \{\vec{t} \in \mathcal{R}^{n-\ell} \mid \sum_{i\in[\ell+1,n]} t_i = 1$ and each $t_i \geq 0\}$ to be our 1-norm unit ball.

OR

Def^n: Let $B = \{\vec{t} \in \mathcal{R}^{n} \mid \sum_i t_i = 1,\ L(\vec{t}) \overset{\text{def}}{=} \sum_i \frac{1}{i} t_i \geq L = 1 - \frac{1}{2\ell},$ and each $t_i \geq 0\}$ to be our 1-norm unit ball.

OR $t_{\ell+1} \geq .5$.

Note that B is a closed, bounded, connected, and without holes, and hence Brouwer's fixed point theorem works for mappings within this space.

In order to use the fixed point theorem, our goal is to define a mapping $A : B \Rightarrow B$. Fix a vector $\vec{t} \in B$. Knowing $\vec{t}$, let $\rho_{\langle i,j\rangle}$ be the allocations specified by Alg. Let $w_i = \sum_{j\geq i} \rho_{\langle i,j\rangle} t_j$ be the work completed on J_i by time $\sum_{i\in[\ell+1,n]} t_i = 1$.

In an attempt to satisfy condition[1], consider changing the t_i to $t'_i = (1+\epsilon)w_i$. Note that this does not actually satisfy condition[1] because changing the t_i directly changes the w_i, but even more seriously it also may cause the algorithm to change the $\rho_{\langle i,j\rangle}$ which again changes the w_i.

Define $E \overset{\text{def}}{=} 1 - \sum_{i\in[\ell+1,n]} t'_i$ to be the extra amount that needs to be added to the t'. Split this some how (????) into $E = E_{\ell+1} + E_n$. Defining the mapping A to be $A(\vec{t})_{\ell+1} = t'_{\ell+1} + E_{\ell+1}$, $A(\vec{t})_i = t'_i$ for $i \in [_{\ell+2}, n-1]$, and $A(\vec{t})_n = t'_n + E_n$. The ball B is closed under the mapping A, i.e. that $A(\vec{t}) \in B$ (????). By construction, $\sum_{i\in[\ell+1,n]} A(\vec{t})_i = 1$. $A(\vec{t})_i \geq 0$, because $\rho_{\langle i,j\rangle} \geq 0$ and (???) proves that $E \geq 0$.

Because B is a closed, bounded, connected, and without holes, and because the mapping $A : B \Rightarrow B$ is closed and continuous, Brouwer's fixed point theorem gives us that there exists a fixed point $\vec{t}$ such that $A(\vec{t}) = \vec{t}$. A quick check will show $\vec{t}$ satisfies constraints[0,6]. For example, constraint[1] is satisfied because for each i, $w_i = \frac{1}{1+\epsilon} t'_i \leq \frac{1}{1+\epsilon} A(\vec{t})_i = \frac{1}{1+\epsilon} t_i$. In fact, all of these inequality will be tight except for $i = n$. □

12 Loss

Lemma 13. *$E = (1+\epsilon)Loss - \epsilon \geq 0$.*

Proof. Alg has speed $\sum_{i\geq j} \rho_{\langle i,j\rangle} = 1$ and hence its total amount of work done during the first $\sum_{i\in[\ell+1,n]} t_i = 1$ time units is $1\times 1 = 1$. We already defined, $w_i \overset{\text{def}}{=} \sum_{j\geq i} \rho_{\langle i,j\rangle} t_j$ is the amount of this work done on job J_i. Similarly, define $Loss \overset{\text{def}}{=} \sum_{j\in[\ell+1,n]} \sum_{i\in[1,\ell]} \rho_{\langle i,j\rangle} t_j$ to be the amount of these resources that the algorithm "wastes" on the first ℓ jobs. It follows that $\sum_{j\in[\ell+1,n]} w_i = 1 - Loss$, that $\sum_{i\in[\ell+1,n]} t'_i = \sum_{i\in[\ell+1,n]}(1+\epsilon)w_i = (1+\epsilon)\,[1 - Loss]$, and that $E \overset{\text{def}}{=} 1 - \sum_{i\in[\ell+1,n]} t'_i = 1 - (1+\epsilon)\,[1 - Loss] = (1+\epsilon)Loss - \epsilon$. □

Lemma 14. *$Loss_j \geq 1 - (1-\frac{\beta}{2})\frac{j}{\ell}$: assume all at zero or all at $\ell+1$*
$Loss_j \geq 2[1 - (1-\frac{\beta}{2})\frac{j}{\ell}]$: assume flat within ℓ or all at $\ell+1$
$Loss_j \geq \frac{\ell}{j} - (1-\beta)$: assume flat within ℓ and flat within $(j-\ell)$
Becomes zero when $\ell = (1-\frac{\beta}{2})j$ or $\ell = (1-\beta)j$. When $\ell = (1-\frac{\beta}{4})j$, $Loss_j \geq \frac{\beta}{4}$ or $\frac{\beta}{2}$ or $\frac{3\beta}{4}$.

13 Ultimate Problems

The details change but the core problem is the same. We need $Loss \geq \epsilon$, (1) it is why Alg does poorly, (2) We need $E = (1+\epsilon)Loss - \epsilon \geq 0$ to keep the t_n positive.

Whether the loss is to old jobs or new jobs the key period is t_1. (In fact old=new for t_1.) The loss rate is the highest there, at a staggering $\frac{\ell}{\ell+1}$. Hence, the way to increase $Loss$ is to increase t_1. This is done in two ways: (1) Show the alg is working on J_1 because $t'_1 = \sum_j \rho_{\langle 1,j\rangle} t_j$. A key problem where is that $\rho_{\langle 1,j\rangle} t_j$ for $j \neq \ell+1$ may be zero, at first because t_j may (but shouldn't) be zero and later because $\rho_{\langle 1,j\rangle}$ may (likely will) be zero. (2) Give it more of the E to E_1. To keep t_1 from shrinking (without the help of t_2), it needs $E_1 = (1-\delta)Loss$ for $\delta = \frac{\epsilon}{\ell}$. This leaves only $E_2 = (\epsilon - \delta)Loss - \epsilon$, which seems to be zero or negative. In Section **??**, t_1 was encouraged to grow. The danger is it becoming one, making all other $t_j = 0$. This is a fine fixed point where everything is balanced. This is one reason things are close but not quite working. Section **??** tried to keep t_1 (or $L = \sum_i \frac{t_i}{i}$) in perfect balance. It found $E_n = (\epsilon - \delta)Loss - \epsilon$ negative, which is odd because if E_1 is given less so it does not increase then why is E_n smaller?

Lemma 15. $A(\vec{t})_1 \geq t_1$.

Proof. Assume Equi during t_1. $A(\vec{t})_1 = t'_1 E_1 = (1+\epsilon)w_1 + (1-\frac{\epsilon}{\ell})Loss = (1+\epsilon)\sum_{j\geq 1}\rho_{\langle 1,j\rangle}t_j + (1-\frac{\epsilon}{\ell})\sum_j \ell\rho_{\langle j,j\rangle}t_j \geq (bigloss???)\ \ (1+\epsilon)\rho_{\langle 1,1\rangle}t_1 + (\ell-\epsilon)\rho_{\langle 1,1\rangle}t_1 = (\ell+1)(\frac{1}{\ell+1}t_1 = t_1$. □

Note, you would think adding increasing the number ℓ of waste jobs (either old or new) would increase $Loss$. It does. But it deceases $t'_1 \approx \rho_{\langle 1,1\rangle}$ by the same amount.

Then $Loss = \sum_j \sum_{i\in[1,\ell]} \rho_{\langle i,j\rangle}t_j \geq (bigloss??) \geq \frac{\ell}{\ell+1}t_1$.

$E_n = (\frac{\epsilon}{\ell}+\epsilon)Loss - \epsilon = (\frac{\epsilon}{\ell}+\epsilon)\frac{\ell}{\ell+1}t_1 - \epsilon = \epsilon(1+\ell)\frac{1}{\ell+1}t_1 - \epsilon = \epsilon(t_1 - 1) \leq 0$!!!

The same is true of not EQUI during $t_{\ell+1}$.

Maybe we should look to giving some to E_2 to guarantee that t_2 is big so that it can give to t_1.

14 Try to increase $\sum_i \frac{1}{i}t_i$.

The bug in this is outlined above in Ultimate Problems.

Split the amount E into $E_1 = (1-\frac{\epsilon}{\ell})Loss$ and $E_n = (\frac{\epsilon}{\ell}+\epsilon)Loss - \epsilon$.

Lemma 16. $L(A(\vec{t})) \geq L(\vec{t})$.

Proof. $L(A(\vec{t})) = \sum_i \frac{1}{i}A(\vec{t})_i = \left[\sum_i \frac{1}{i}t'_i\right] + \frac{1}{1}E_1 + \frac{1}{n}E_n \geq \left[\sum_i \frac{1}{i}[(1+\epsilon)w_i]\right] + (1-\frac{\epsilon}{\ell})Loss = \left[\sum_i \frac{1+\epsilon}{i}\left[\sum_{j\geq i}\rho_{\langle i,j\rangle}t_j\right]\right] + (1-\frac{\epsilon}{\ell})\left[\sum_j \ell\rho_{\langle j,j\rangle}t_j\right] = \sum_j \left[\left[\sum_{i\leq j}\frac{1+\epsilon}{i}\rho_{\langle i,j\rangle}\right] + \left[(1-\frac{\epsilon}{\ell})\ell\rho_{\langle j,j\rangle}\right]\right]t_j \stackrel{\text{def}}{=} \sum_j [Q_j]\,t_j$. $L(\vec{t}) = \sum_i \frac{1}{j}t_j$. Hence, to prove the lemma, it is sufficient to prove that $\forall j,\ Q_j \geq \frac{1}{j}$.

Note that the amount $\sum_{i\leq j}\frac{1+\epsilon}{i}\rho_{\langle i,j\rangle}$ depends on how the algorithm distributes its resources to the j parallelizable jobs during the time period $t \in [r_j, r_j + t_j]$. It is clearly minimized by moving these resources forward to the jobs with the larger index i, i.e. those arriving later. However, because the algorithm *favors the most recent job*, $\rho_{\langle i,j\rangle} \leq \rho_{\langle j,j\rangle}$. It follows that $\sum_{i\leq j}\frac{1+\epsilon}{i}\rho_{\langle i,j\rangle}$ is minimized by choosing some index r_j and having $\rho_{\langle i,j\rangle} = 0$ for $i \in [1, j{-}r_j]$ and $\rho_{\langle i,j\rangle} = \rho_{\langle j,j\rangle}$ for $i \in [j{-}r_j{+}1, j]$. This gives r_j parallel and ℓ sequential jobs receiving this same allocation. Being speed one, gives $(r_j+\ell)\rho_{\langle j,j\rangle} = 1$ or $r_j = \frac{1}{\rho_{\langle j,j\rangle}} - \ell$. This gives $Q_j = \left[\sum_{i\in[j-r_j+1,j]}\frac{1+\epsilon}{i}\rho_{\langle j,j\rangle}\right] + \left[(1-\frac{\epsilon}{\ell})\ell\rho_{\langle j,j\rangle}\right] = \left[\left[\sum_{i\in[j-r_j+1,j]}\frac{1+\epsilon}{i}\right] + (\ell-\epsilon)\right]\rho_{\langle j,j\rangle}$.

It is true that decreasing $\rho_{\langle j,j\rangle}$, increases r_j, which increases the harmonic sum, however, calculus will show that this increase will be overshadowed by the fact that this harmonic sum is being multiplied by $\rho_{\langle j,j\rangle}$. Hence, this expression is minimized by setting $\rho_{\langle j,j\rangle}$ to be as small as possible. This is done by running *Equi*, i.e. $\rho_{\langle j,j\rangle} = \rho_{\langle i,j\rangle} = \frac{1}{j+\ell}$. This gives $Q_j = \left[\left[\sum_{i\in[1,j]} \frac{1+\epsilon}{i}\right] + (\ell - \epsilon)\right] \frac{1}{j+\ell} \geq [1+\ell]\frac{1}{j+\ell} \geq \frac{1}{j}$. □

Lemma 17. *$\sum_i it_i \geq \frac{\epsilon}{3\ell}n$.*

Proof. $\sum_i it_i = \sum_i iA(\vec{t})_i = [\sum_i it'_i] + 1E_1 + nE_n \geq nE_n$. Recall $E_n = (\frac{\epsilon}{\ell}+\epsilon)Loss - \epsilon = (\frac{\epsilon}{\ell}+\epsilon)(1-\frac{1}{2\ell}) - \epsilon \geq \frac{\epsilon}{3\ell}$. □

Bug!! $Loss = \sum_j \sum_{i\in[1,\ell]} \rho_{\langle i,j\rangle} t_j \geq (bigloss??) \geq \frac{\ell}{\ell+1}t_1$. This is smaller than L.

15 Try to Fix $\sum_i \frac{1}{i}t_i$.

The bug in this is outlined above in Ultimate Problems.

Define two *extra amounts* to be

$$\begin{aligned} E_1 &\overset{\text{def}}{=} L - L(\vec{t'}) - \tfrac{1}{n}E_n \\ E_n &\overset{\text{def}}{=} \frac{1}{1-\frac{1}{n}}[L(\vec{t'}) + (1+\epsilon)Loss(\vec{\rho},\vec{t}) - (L+\epsilon)] \end{aligned}$$

Lemma 18. *$\sum_i A(\vec{t})_i = 1$*

Proof.

$$\begin{aligned} \sum_i A(\vec{t})_i &= \sum_i t'_i + E_1 + E_n \;=\; \sum_i t'_i + \left[L - L(\vec{t'}) - \tfrac{1}{n}E_n\right] + E_n \;=\; \sum_i t'_i + L - L(\vec{t'}) + (1-\tfrac{1}{n})E_n \\ &= \sum_i [(1+\epsilon)w_i] + L - L(\vec{t'}) + \left[L(\vec{t'}) + (1+\epsilon)Loss(\vec{\rho},\vec{t}) - (L+\epsilon)\right] \\ &= \left[(1+\epsilon)(1 - Loss(\vec{\rho},\vec{t}))\right] + (1+\epsilon)Loss(\vec{\rho},\vec{t}) - \epsilon \;=\; 1 \end{aligned}$$

□

Lemma 19. *$L(A(\vec{t})) = L$.*

Proof.

$$\begin{aligned} L(A(\vec{t})) &= \sum_i \tfrac{1}{i}A(\vec{t})_i \;=\; \left[\sum_i \tfrac{1}{i}t'_i\right] + \tfrac{1}{1}E_1 + \tfrac{1}{n}E_n \\ &= \left[L(\vec{t'})\right] + \left[L - L(\vec{t'}) - \tfrac{1}{n}E_n\right] + \tfrac{1}{n}E_n \;=\; L \end{aligned}$$

□

Lemma 20. *$R(A(\vec{t})) = ?$.*

Proof.

$$\begin{aligned}
\tfrac{1}{n}R(A(\vec{t})) &= \tfrac{1}{n}\sum_i iA(\vec{t})_i \;=\; \tfrac{1}{n}\left[\left[\sum_i it'_i\right] + 1E_1 + nE_n\right] \\
&\geq \left[\sum_i \tfrac{i}{n}t'_i\right] + \left[L(\vec{t'}) + (1+\epsilon)Loss(\vec{\rho},\vec{t}) - (L+\epsilon)\right] \\
&= \left[\sum_i (\tfrac{i}{n}+\tfrac{1}{i})t'_i\right] + (1+\epsilon)\left[\sum_j \ell\rho_{\langle j,j\rangle}t_j\right] - (L+\epsilon) \\
&= \left[\sum_i (\tfrac{i}{n}+\tfrac{1}{i})\left[(1+\epsilon)\sum_{j\geq i}\rho_{\langle i,j\rangle}t_j\right]\right] + (1+\epsilon)\left[\sum_j \ell\rho_{\langle j,j\rangle}t_j\right] - (L+\epsilon) \\
&= \left[(1+\epsilon)\sum_j \left[\left[\sum_{i\leq j}(\tfrac{i}{n}+\tfrac{1}{i})\rho_{\langle i,j\rangle}\right] + \ell\rho_{\langle j,j\rangle}\right]t_j\right] - (L+\epsilon)
\end{aligned}$$

We will drop the $\frac{i}{n}$. The amount $Q_j \overset{\text{def}}{=} \left[\sum_{i\leq j}\frac{1}{i}\rho_{\langle i,j\rangle}\right] + \ell\rho_{\langle j,j\rangle}$ depends on how the algorithm distributes its resources to the j parallelizable jobs during the time period $t \in [r_j, r_j + t_j]$. It is clearly minimized by moving these resources forward to the jobs with the larger index i, i.e. those arriving later. However, because the algorithm *favors the most recent job*, $\rho_{\langle i,j\rangle} \leq \rho_{\langle j,j\rangle}$. It follows that Q_j is minimized by choosing some index r_j and having $\rho_{\langle i,j\rangle} = 0$ for $i \in [1, j-r_j]$ and $\rho_{\langle i,j\rangle} = \rho_{\langle j,j\rangle}$ for $i \in [j-r_j+1, j]$. This gives r_j parallel and ℓ sequential jobs receiving this same allocation. Being speed one, gives $(r_j+\ell)\rho_{\langle j,j\rangle} = 1$ or $r_j = \frac{1}{\rho_{\langle j,j\rangle}} - \ell$. This gives $Q_j = \left[\sum_{i\in[j-r_j+1,j]}\frac{1}{i}\rho_{\langle j,j\rangle}\right] + \ell\rho_{\langle j,j\rangle} = \left[\left[\sum_{i\in[j-r_j+1,j]}\frac{1}{i}\right] + \ell\right]\rho_{\langle j,j\rangle}$.

It is true that decreasing $\rho_{\langle j,j\rangle}$, increases r_j, which increases the harmonic sum, however, calculus will show that this increase will be overshadowed by the fact that this harmonic sum is being multiplied by $\rho_{\langle j,j\rangle}$. Hence, this expression is minimized by setting $\rho_{\langle j,j\rangle}$ to be as small as possible. This is done by running *Equi*, i.e. $\rho_{\langle j,j\rangle} = \rho_{\langle i,j\rangle} = \frac{1}{j+\ell}$. This gives $Q_j = \left[\left[\sum_{i\in[1,j]}\frac{1}{i}\right] + \ell\right]\frac{1}{j+\ell} \geq [1+\ell]\frac{1}{j+\ell} \geq \frac{1}{j}$. ** This is tight if when $j = 1$ and does not matter for $j > 1$ if $t_j = 0$. **

$$\begin{aligned}
\tfrac{1}{n}R(A(\vec{t})) &= \left[(1+\epsilon)\sum_j \left[\tfrac{1}{j}\right]t_j\right] - (L+\epsilon) \\
&= (1+\epsilon)L - (L+\epsilon) = \epsilon L - \epsilon = neg
\end{aligned}$$

□

16 Increasing $F_J(\vec{t}) \overset{\text{def}}{=} \sum_{i\in[\ell+1,J]} it_i$

Proof.

$$\begin{aligned}
Loss &\overset{\text{def}}{=} \sum_{j\in[\ell+1,n]}\sum_{i\in[1,\ell]} \rho_{\langle i,j\rangle} t_j \;\geq\; \tfrac{1}{\ell}\sum_{j\in[\ell+1,J]}\sum_{i\in[1,\ell]} i\rho_{\langle i,j\rangle} t_j \\
&\;\text{Maybe can get an extra factor of 2} \\
F_J(\vec{t'}) &\overset{\text{def}}{=} \sum_{i\in[\ell+1,J]} it'_i \;=\; \sum_{i\in[\ell+1,J]} i(1+\epsilon)w_i \;\geq\; (1+\epsilon)\sum_{i\in[\ell+1,J]} i\left[\sum_{j\in[i,J]}\rho_{\langle i,j\rangle}t_j\right] \\
&= (1+\epsilon)\sum_{j\in[\ell+1,J]}\sum_{i\in[\ell+1,j]} i\rho_{\langle i,j\rangle}t_j \\
F_J(\vec{t'}) + (1+\epsilon)\ell Loss &= (1+\epsilon)\sum_{j\in[\ell+1,J]}\left[\sum_{i\in[1,j]} i\rho_{\langle i,j\rangle}\right]t_j \;=\; (1+\epsilon)\sum_{j\in[\ell+1,J]}\left[(1-\tfrac{\beta}{2})j\right]t_j \\
&= (1+\epsilon)(1-\tfrac{\beta}{2})F_J(\vec{t})
\end{aligned}$$

□

17 Changing β_j

Also does not work.

Lemma 21. *The minimum for the amount amount $\sum_{j\in[\ell+1,n]} jt_j$ increases with β_j.*

Proof.

$$\begin{aligned}
R(\vec{w}) &\overset{\text{def}}{=} \sum_{i\in[\ell+1,n]} iw_i \;=\; \sum_{i\in[\ell+1,n]} i\sum_{j\geq i}\rho_{\langle i,j\rangle}t_j \;=\; \sum_{j\in[\ell+1,n]}\left[\sum_{i\in[\ell+1,j]} i\rho_{\langle i,j\rangle}\right]t_j \\
R(Loss) &\overset{\text{def}}{=} \sum_{j\in[\ell+1,n]}\sum_{i\in[1,\ell]} i\rho_{\langle i,j\rangle}t_j \;=\; o\left(\sum_{j\in[\ell+1,n]}\sum_{i\in[1,\ell]} n\rho_{\langle i,j\rangle}t_j\right) \;=\; o\left(nLoss\right) \\
R(\vec{w}) + R(Loss) &= \sum_{j\in[\ell+1,n]}\left[\sum_{i\in[1,j]} i\rho_{\langle i,j\rangle}\right]t_j \;=\; \sum_{j\in[\ell+1,n]}\left[j(1-\tfrac{\beta_j}{2})\right]t_j \\
&\leq (1-\tfrac{\beta}{2})\sum_{j\in[\ell+1,n]} jt_j \;\overset{\text{def}}{=}\; (1-\tfrac{\beta}{2})R(\vec{t}) \\
R(\vec{t'}) &\overset{\text{def}}{=} \sum_{i\in[\ell+1,n]} it'_i \;=\; \sum_{i\in[\ell+1,n]} i(1+\epsilon)w_i \;=\; (1+\epsilon)R(\vec{w}) \\
R(A(\vec{t})) &\overset{\text{def}}{=} \sum_{i\in[\ell+1,n]} iA(\vec{t})_i \;\leq\; \sum_{i\in[\ell+1,n]} it'_i \;+\; nE \\
&= (1+\epsilon)\left[(1-\tfrac{\beta}{2})R(\vec{t}) - R(Loss)\right] \;+\; n\left[(1+\epsilon)Loss - \epsilon\right] \\
&= (1+\epsilon)\left[(1-\tfrac{\beta}{2})R(\vec{t}) + (1-o(1))nLoss\right] \;-\; \epsilon n
\end{aligned}$$

Increasing β_j increase amount allocated to earlier jobs, which increases *Loss*.

Ooooooops. It decreases $1 - \frac{\beta_j}{2}$.

□

18 Absurd Kludge

Initially $\ell + 1$ jobs arrive at time zero. One can assume that job J_{l+1} is given at least its share $\rho_{\langle \ell+1, \ell+1\rangle} = \frac{1}{\ell+1}$ of the resources. What if this was not the case?

This does not work as well.

Lemma 22. $A(\vec{t})_{\ell+1} = t_{\ell+1} = \frac{1}{2}$.

Proof.

$$\begin{aligned}
t'_{\ell+1} &= (1+\epsilon)w_{\ell+1} \;=\; (1+\epsilon)\sum_{j\geq 1}\rho_{\langle \ell+1,j\rangle}t_j \;\geq\; (1+\epsilon)\rho t_{\ell+1}.\\
\sum_{i\in[1,\ell+1]}\rho_{\langle i,\ell+1\rangle} &= 1. \qquad\qquad \rho \overset{\text{def}}{=} \rho_{\langle \ell+1,\ell+1\rangle}.\\
Loss &= \sum_{j\in[\ell+1,n]}\sum_{i\in[1,\ell]}\rho_{\langle i,j\rangle}t_j \;\geq\; \sum_{i\in[1,\ell]}\rho_{\langle i,\ell+1\rangle}t_{\ell+1} \;=\; (1-\rho)t_{\ell+1}.\\
E_{\ell+1} &= (1-\delta)Loss. \qquad \delta = \tfrac{\epsilon\rho}{1-\rho}.\\
A(\vec{t})_{\ell+1} &= t'_{\ell+1} + E_{\ell+1} \;\geq\; \left[(1+\epsilon)\rho + (1-\delta)(1-\rho)\right]t_{\ell+1}\\
&= \left[(1+\epsilon)\rho + (1-\tfrac{\epsilon\rho}{1-\rho})(1-\rho)\right]t_{\ell+1}\\
&= \left[\rho+\epsilon\rho+(1-\rho)-\epsilon\rho\right]t_{\ell+1} \;=\; t_{\ell+1}
\end{aligned}$$

□

Lemma 23. *If* $\rho_{\langle \ell+1,\ell+1\rangle} \leq \frac{\beta}{\ell+1}$*, then* $\sum_{i\in[\ell+1,n]} iA(\vec{t})_i \geq ??$.

Proof.

$$\begin{aligned}
\sum_i iA(\vec{t})_i &= \left[\sum_i it'_i\right] + 1E_1 + nE_n \;\geq\; nE_n.\\
E_n &= (\delta+\epsilon)Loss - e \;=\; (\tfrac{\epsilon\rho}{1-\rho}+\epsilon)(1-\rho)t_{\ell+1} - \epsilon\\
&= (\epsilon\rho + \epsilon(1-\rho))t_{\ell+1} - \epsilon \;=\; \epsilon t_{\ell+1} - \epsilon \leq 0
\end{aligned}$$

□