A Better Algorithm For an Ancient Scheduling Problem

David R. Karger * Steven J. Phillips * Eric Torng *

Department of Computer Science
Stanford University
Stanford, CA 94305-2140

Abstract

One of the oldest and simplest variants of multiprocessor scheduling is the on-line scheduling problem studied by Graham in 1966. In this problem, the jobs arrive on-line and must be scheduled non-preemptively on m identical machines so as to minimize the makespan. The size of a job is known on arrival. Graham proved that the List Processing Algorithm which assigns each job to the currently least loaded machine has competitive ratio $(2 - 1/m)$. Recently algorithms with smaller competitive ratios than List Processing have been discovered, culminating in Bartal, Fiat, Karloff, and Vohra's construction of an algorithm with competitive ratio bounded away from 2. Their algorithm has a competitive ratio of at most $(2 - 1/70) \approx 1.986$ for all m; hence for $m > 70$, their algorithm is provably better than List Processing.

We present a more natural algorithm that outperforms List Processing for any $m \geq 6$ and has a competitive ratio of at most 1.945 for all m, which is significantly closer to the best known lower bound of 1.837 for the problem. We show that our analysis of the algorithm is almost tight by presenting a lower bound of 1.9378 on the algorithm's competitive ratio for large m.

1 Introduction

Scheduling n jobs on m machines is one of the most widely studied problems in computer science. One of its earliest and simplest variants is the on-line scheduling problem introduced by Graham [5] in 1966. The m machines are identical, and the n nonpreemptable, single task jobs are all independent. Job i has size $J_i > 0$. The jobs arrive one by one, and each job must be immediately and irrevocably scheduled without knowledge of later jobs. The size of a job is known on arrival, and the jobs are executed only after the scheduling is completed. The goal is to minimize the makespan — the completion time of the last job to finish.

This problem is also referred to as *load-balancing*. For example, the "machines" can be communication channels, and the "jobs" can be requests for communication bandwidth. When a customer requests bandwidth, a channel must be chosen on which the required bandwidth is immediately and permanently made available to the customer. Under this interpretation, the goal is to minimize the maximum load on any channel.

Because of the online nature of the problem, the performance of a scheduler is measured by its *competitive ratio*. For a job sequence σ, let $A(\sigma)$ denote the makespan of algorithm A's schedule, and let $OPT(\sigma)$ denote the minimum makespan of all m-machine schedules for σ. The competitive ratio of A is defined by

$$C_A \stackrel{\text{def}}{=} \sup_{\sigma} \frac{A(\sigma)}{OPT(\sigma)},$$

where the supremum is over all nonempty job sequences.

The natural question is, how small can C_A be? Graham showed that the List Processing Algorithm (List for brevity) which assigns each job to the least loaded machine has competitive ratio exactly $2 - \frac{1}{m}$. Starting in 1991, several algorithms were developed [4, 6] with better competitive ratios than List, and recently Bartal, Fiat, Karloff, and Vohra [1] gave an algorithm with a competitive ratio of $2 - \frac{1}{70} \approx 1.986$. Faigle, Kern, and Turan [3] proved lower bounds on the achievable deterministic competitive ratio of $2 - \frac{1}{m}$ for $m = 2$ and 3 and $1 + \frac{1}{\sqrt{2}}$ for $m \geq 4$ (so List is optimal for $m = 2$ and 3). Bartal, Karloff, and Rabani [2] improved this lower bound for $m \geq 4$ to $1 + \frac{1}{\sqrt{2}} + \epsilon_m$ which becomes 1.837 for m large enough.

This paper presents a deterministic algorithm, ALG_α, that is more natural than the algorithm of Bartal et al. The algorithm ALG_α uses a parameter α that affects its behavior; the best choice of α depends on m. For $m \geq 6$, there is an α such that ALG_α outperforms List; furthermore, $ALG_{1.945}$ has a competitive ratio of 1.945 for all m. Figure 1 shows how ALG_α (under the best choice of α) compares to List, Bartal et al's algorithm, and the asymptotic lower bound.

We show that our analysis of ALG_α is almost tight by presenting a lower bound of 1.9378 on the

*Supported by NSF Grant CCR-9010517, NSF Young Investigator Award CCR-9357849, and grants from Mitsubishi Corporation and OTL.

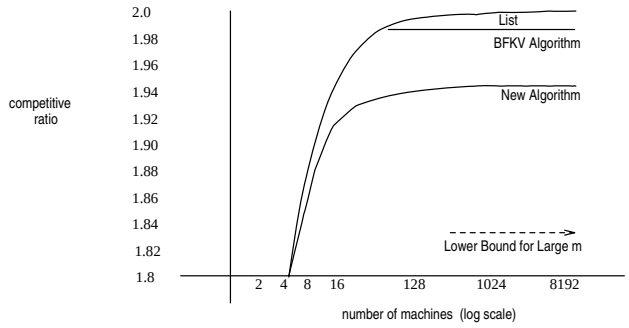

Figure 1: Comparison of our algorithm to List, Bartal et al's algorithm, and the asymptotic lower bound

competitive ratio of ALG_α (for each choice of α) for large m.

2 Definition of ALG_α

In order to do better than List, we must see what List does wrong in its worst case, a sequence of $m(m-1)$ jobs of size 1 followed by one job of size m. List assigns the large job to a machine that already has $m-1$ small jobs while the optimal schedule assigns the large job to its own machine. The problem is that List keeps the schedule so flat that when the large job arrives it has to go on a relatively tall machine. The algorithm ALG_α strives to maintain an imbalance in the processor loads so that a large job can always be assigned to a relatively short machine. However, it cannot create too large an imbalance because this would immediately imply a poor competitive ratio.

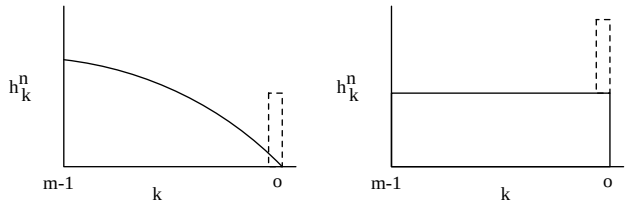

Figure 2: Scheduling many small jobs, then one big job

Define the *height* (or load) of a machine to be the sum of the sizes of all jobs currently assigned to it.

DEFINITION 2.1. *Time t is the time just before the t^{th} job is scheduled. At time t: M_i^t is the $(i+1)^{st}$ shortest machine (e.g. M_0^t is the shortest machine), h_i^t is the height of M_i^t $(0 \le i < m)$, A_i^t is the average height of the i shortest machines, $1 \le i \le m$, and $A_0^t = \infty$.*

Note that A_m^t is the average height of all m machines at time t and is thus a lower bound on OPT.

DEFINITION 2.2. *The algorithm ALG_α works as follows: when job t arrives, it is placed on the tallest machine M_k^t such that $h_k^t + J_t \le \alpha A_k^t$.*

The parameter α determines the degree of imbalance that ALG_α tries to maintain. Note that with the above definition of A_0^t, any very tall job will be placed on the shortest machine. The algorithm of Bartal et al is a discrete version of ALG_α where a job can only be placed on machine M_0^t (the smallest machine) or machine $M_{\delta m}^t, \delta \approx 0.445$.

The reader may wonder why we use the average height of the shorter machines *before* adding the new job rather than the average height of the shorter machines *after* adding the new job (i.e. comparing $h_k^t + J_t$ to $\alpha A_{k'}^{t+1}$ where k' is the rank of machine M_k^t with job t at time $t+1$). Furthermore, why not compare the new height of the machine to the average height of all the machines (i.e. compare $h_k^t + J_t$ to αA_m^{t+1})? Both of these are better approximations to $OPT(\sigma)$, so they would seem to be better choices. However, neither algorithm is better than $(2 - \frac{1}{m})$-competitive. In particular, when these algorithms schedule a long sequence of equal-sized jobs, they distribute almost all of the jobs evenly on slightly more than half the machines while the rest of the machines get only a small number of jobs. An adversary can then insert approximately $\frac{m}{2}$ large jobs to create a completely flat schedule, after which a single large job makes the makespan $2 - \frac{1}{m}$ times the optimal makespan.

In the next two sections we prove the following theorem.

THEOREM 2.1. *For each $m \ge 6$, there exists some α such that ALG_α outperforms List, and for $\alpha \ge 1.945$, ALG_α is α-competitive for all m.*

3 Upper Bound: Fixed m

Fix $m \ge 6$, and fix α, $1 \le \alpha < 2$. This section shows how to compute an upper bound on the competitive ratio of ALG_α on m machines, by induction on the length of the job sequence. Let σ be a job sequence of length n, normalized so that the total size of all jobs in σ is m (and thus $A_m^{n+1} = 1$). Let σ_{n-1} be the first $n-1$ jobs in σ, and assume inductively that $ALG_\alpha(\sigma_{n-1}) \le \alpha OPT(\sigma_{n-1})$. The rest of this section is devoted to showing that $ALG_\alpha(\sigma) \le \alpha OPT(\sigma)$.

The following lower bounds on $OPT(\sigma)$ will be used.

FACT 3.1. *The following quantities are lower bounds on $OPT(\sigma)$:*

1. *$\frac{1}{m}$ times the total size of all jobs in σ $(= 1)$.*
2. *The largest job in σ $(\ge J_n)$.*
3. *Twice the size of the $(m+1)^{st}$ largest job in σ.*

If $ALG_\alpha(\sigma) \neq h_0^n + J_n$, then the definition of ALG_α and the inductive assumption easily imply $ALG_\alpha(\sigma) \leq \alpha OPT(\sigma)$. If $ALG_\alpha(\sigma) \leq \alpha$ or $ALG_\alpha(\sigma) \leq \alpha J_n$, then Fact 3.1 (part (1) or (2) respectively) implies $ALG_\alpha(\sigma) \leq \alpha OPT(\sigma)$. Therefore, for the rest of this section, assume

$$ALG_\alpha(\sigma) = h_0^n + J_n, \tag{3.1}$$

$$ALG_\alpha(\sigma) \geq \alpha, \tag{3.2}$$

$$ALG_\alpha(\sigma) \geq \alpha J_n. \tag{3.3}$$

We prove that for $m \geq 6$, the $(m+1)^{st}$ largest job in σ has size at least

$$\frac{1}{2}\frac{ALG_\alpha(\sigma)}{\alpha} \tag{3.4}$$

(for a suitable choice of α), which combined with Fact 3.1 part (3) proves $ALG_\alpha(\sigma) \leq \alpha OPT(\sigma)$.

DEFINITION 3.1. *Define β by $1-\beta = h_0^n$. Define $\epsilon = 2-\alpha$. Define $b = \frac{1}{2}\frac{1-\beta}{1-\epsilon}$. Define $a = 1-\beta-b$.*

DEFINITION 3.2. *A machine is* tall *if it has height at least $1-\beta$; otherwise it is* short. *Let s be the current number of short machines (so A_s^t is the average height of the short machines at time t). A job is* large *if it has size $\geq b$.*

DEFINITION 3.3. *An* elevating job *raises a machine from being short to being tall, and we say that an elevating job* elevates *the machine it is placed on.*

Each machine is tall by time n, so there are m elevating jobs in σ. We will prove that the last job is large and that all the elevating jobs are large. Lemma 3.2 below shows that a large job has size at least $\frac{1}{2}\frac{ALG_\alpha(\sigma)}{\alpha}$, as required by (3.4) above.

The sequence σ is divided into two phases. In the first phase, $A_s^t < a$, and each elevating job in this phase must go on the smallest machine and is therefore large. A_s^t increases monotonically with t until we reach the second phase (called the kickstart) when $A_s^t \geq a$.

The kickstart is analyzed in reverse: starting with the last elevating job in σ and moving backwards in time, we show that each elevating job is large. Eventually a time t is reached where so much processing time is taken by the jobs in the tall machines and the later elevating jobs that we must have $A_s^t < a$. Thus the first phase of the proof applies to all preceding times.

3.1 Large jobs

LEMMA 3.1. *β satisfies $0 < \beta \leq \epsilon$.*

Proof. The fact that $\beta > 0$ follows from $1-\beta = h_0^n \leq A_m^n < A_m^{n+1} = 1$. To see that $\beta \leq \epsilon$, we have from (3.1) (3.2) and (3.3) above that $h_0^n + J_n \geq \alpha J_n$ and $h_0^n + J_n \geq \alpha$. The first inequality gives

$$J_n \leq \frac{h_0^n}{1-\epsilon}. \tag{3.5}$$

Substituting this upper bound for J_n into the second inequality gives

$$h_0^n(1 + \frac{1}{1-\epsilon}) \geq \alpha,$$

which simplifies to $1-\beta = h_0^n \geq 1-\epsilon$.

LEMMA 3.2. $b \geq \frac{1}{2}\frac{ALG_\alpha(\sigma)}{\alpha}$

Proof. Equation (3.5) above gives $ALG_\alpha(\sigma) = h_0^n + J_n \leq h_0^n(1 + \frac{1}{1-\epsilon}) = \frac{\alpha(1-\beta)}{1-\epsilon}$. The lemma follows from the definition $b = \frac{1}{2}\frac{1-\beta}{1-\epsilon}$.

Thus a large job has size at least $\frac{1}{2}\frac{ALG_\alpha(\sigma)}{\alpha}$.

LEMMA 3.3. *If $\epsilon \leq 1 - \frac{1}{\sqrt{2}} \approx .29$, then job n is large.*

Proof. Job n raises a machine from height $1-\beta$ to at least $\alpha = 2-\epsilon$, so $J_n \geq 1-\epsilon+\beta$. Job n is large whenever $1-\epsilon+\beta \geq \frac{1}{2}\frac{1-\beta}{1-\epsilon}$. Setting $\beta = 0$, the worst case, this simplifies to $\epsilon \leq 1 - \frac{1}{\sqrt{2}} \approx .29$.

3.2 The First Phase

This section proves that σ can be broken into two phases, such that $A_s^t < a$ during the first phase, $A_s^t \geq a$ in the second phase, and all elevating jobs in the first phase are large.

LEMMA 3.4. *If job t is an elevating job for machine M_k^t, then $A_k^t \geq a$.*

Proof. Suppose job t goes on machine M_k^t. The rule for ALG_α implies that $\alpha A_k^t \geq h_k^t + J_t$. Since job t is an elevating job, we have $h_k^t + J_t \geq 1-\beta$. By transitivity, $\alpha A_k^t \geq 1-\beta$, so $A_k^t \geq \frac{1-\beta}{\alpha} > a$.

LEMMA 3.5. *If job t is an elevating job and $A_s^t < a$, then job t goes on the shortest machine M_0^t and is large.*

Proof. For all $0 < i < s$, $A_i^t < A_s^t < a$. Therefore, by Lemma 3.4, elevating job t cannot go on any short machine except the shortest, machine M_0^t. Since M_0^t is the shortest machine, $h_0^t \leq A_s^t < a$. Job t is an elevating job, so $h_0^t + J_t \geq 1-\beta$ which implies $J_t > 1-\beta-a = b$. Therefore, job t is large.

LEMMA 3.6. *If $A_s^{t+1} < a$ then $A_s^t \leq A_s^{t+1} < a$.*

Proof.

Case 1: job t is not elevating. The number of short machines is the same at times t and $t+1$. The only difference between time t and time $t+1$ is the absence of job t at time t. Therefore, $A_s^t \leq A_s^{t+1}$.

Case 2: job t is elevating. There is one more short machine at time t than at time $t+1$. We show that job t must be placed on the shortest machine at time t. This implies $A_s^t \leq A_s^{t+1}$, since the average height of the short machines is increased by the removal from consideration of the shortest such machine.

Assume that elevating job t is placed on short machine M_k^t, with $k > 0$. Now $A_k^t = A_k^{t+1}$ (since for $j < k$, $h_j^t = h_j^{t+1}$). But $A_k^{t+1} \leq A_s^{t+1} < a$, and $A_k^t \geq a$ by Lemma 3.4, giving a contradiction. Therefore, job t must go on machine M_0^t.

COROLLARY 3.1. *If $A_s^t < a$, then $A_s^u \leq A_s^t < a$ for all $u < t$. If $A_s^t \geq a$, then $A_s^u \geq a$ for all $u > t$.*

COROLLARY 3.2. *Each elevating job that arrives during the first phase is large.*

3.3 The Kickstart

This section shows that all elevating jobs that arrive during Phase 2 are large. Consider the elevating jobs in reverse order of arrival. Let e_s be the s^{th} elevating job from the last to arrive; say it arrives at time t_s. Given lower bounds on the heights $e_1, \ldots, e_{s-1}$, we derive a lower bound on e_s (note s is the number of short machines at time t_s). We continue until the lower bound on $\sum_{j \leq s} e_j$ is so large that the fact that $A_m^{n+1} = 1$ forces $A_s^{t_s} < a$, proving that time t_s must be in Phase 1. If each lower bound on e_j, $j < s$, exceeds b, then we have proven that all elevating jobs that arrive during Phase 2 are large.

LEMMA 3.7. *Given β, α, and lower bounds for $e_1 \cdots e_{s-1}$, and assuming that $A_s^{t_s} \geq a$, a lower bound for e_s for ALG_α can be computed by solving the following linear program. Minimize e_s subject to*

$$m \geq mA_m^{t_s} + \sum_{j=1}^{s} e_j + 1 - \epsilon + \beta \tag{3.6}$$

$$h_j^{t_s} \geq max(\alpha A_j^{t_s} - e_s, 1 - \beta), \quad s \leq j < m \tag{3.7}$$

$$A_s^{t_s} \geq a \tag{3.8}$$

$$A_j^{t_s} = \frac{1}{j}\sum_{k=0}^{j-1} h_j^{t_s}, \quad 1 \leq j \leq m \tag{3.9}$$

Proof. At any time $0 \leq t \leq n$, the size of the jobs that arrive before t and the size of the jobs that arrive after t must sum to m. The sum of the jobs that arrive before time t_s is at least $mA_m^{t_s}$, while the sum of the jobs that arrive after time t_s is at least $\sum_{j=1}^{s} e_j + J_n$, and $J_n \geq 1 - \epsilon + \beta$ (from Lemma 3.3). This implies equation (3.6). Refer to figure 3. The $m - s$ equations in (3.7) follow from the fact that e_s cannot go on any of the tall machines (because it is an elevating job). Refer to figure 4. Equation (3.8) is part of the hypothesis of the lemma, while equation (3.9) is the definition of $A_j^{t_s}$.

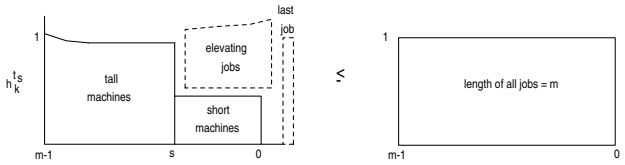

Figure 3: Equation (3.6)

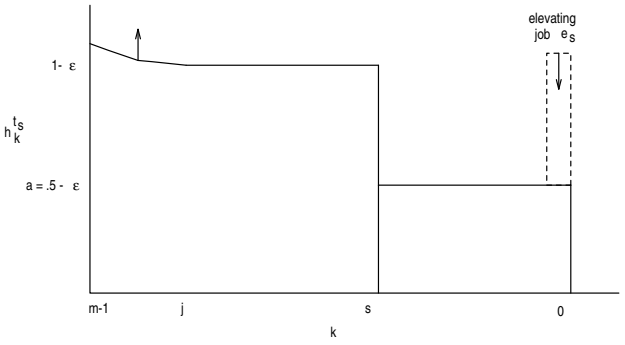

Figure 4: Equation (3.7)

COROLLARY 3.3. *For any m, α, and β, let $e_1, \ldots, e_i, \ldots, e_{s-1}$ be successive solutions to the linear program of Lemma 3.7. If e_j is large for $1 \leq j < s$ and there is no feasible solution to the linear program for e_s, then $A_s^{t_s} < a$ and each elevating job that arrives during Phase 2 is large.*

Proof. The fact that the linear program of Lemma 3.7 for e_s is infeasible implies that the assumed conditions at time t_s cannot hold which means $A_s^{t_s} < a$. Therefore, Phase 1 must last at least until time t_s. We have calculated the minimum values of all later elevating jobs, and, by assumption, they are large. Therefore, all elevating jobs that arrive during Phase 2 must be large.

Note that Corollary 3.3 depends on β. We remove this dependence by showing that the worst value of β for our analysis is $\beta = \epsilon$.

LEMMA 3.8. *Assume that $\epsilon \leq 0.2$. If the conditions of Corollary 3.3 hold (so the kickstart is successful) for $\beta = \epsilon$, then they hold for all β, $0 \leq \beta \leq \epsilon$.*

Proof. Let a superscript on a variable denote the value of that variable with β equal to the superscript (e.g. b^γ is the value of b when $\beta = \gamma$). Lemma 3.8 is proven by showing that if $e_1^\epsilon \cdots e_k^\epsilon > b^\epsilon$ and the linear program for e_{k+1}^ϵ is infeasible, then for some $j \leq k$, we

have $e_1^\gamma \cdots e_j^\gamma > b^\gamma$ and the linear program for e_{j+1}^γ is infeasible. The full proof is included in the appendix.

Thus to determine if ALG_α is α-competitive on m machines, we need to solve the linear programs described in Lemma 3.7 and Corollary 3.3 only for the case $\beta = \epsilon$.

COROLLARY 3.4. *If the conditions of Corollary 3.3 hold for α and m (and $\beta = \epsilon$), so that ALG_α is α-competitive for m machines, then ALG_γ is γ-competitive for m machines for $\alpha \le \gamma \le 2$.*

Proof. This follows from the fact that the constraints of Lemma 3.7 are tighter for larger α.

LEMMA 3.9. *When $6 \le m \le 13000$, there exists an α less than $2 - \frac{1}{m}$ and at most 1.943 such that ALG_α is α-competitive on m machines.*

Proof. We have written code to solve the linear programs of Lemma 3.7 for $6 \le m \le 13000$ and $1 \le s \le m$. It uses binary search to find the smallest α (within a tolerance of .00001) such that the conditions of Corollary 3.3 are satisfied, and hence $ALG_\alpha(\sigma) \le \alpha OPT(\sigma)$. In all cases, $\alpha < 2 - \frac{1}{m}$ and $\alpha \le 1.943$.

4 Upper Bound: Large m

The linear programs of Lemma 3.7 only apply to a specific value of m, and it is impossible to solve them for all m. To give a bound for all large m, we discretize the linear program of Lemma 3.7, effectively regarding groups of machines as having the same height.

Let x be a constant such that $m > x(x-2)$. Let $b = \lceil \frac{m}{x} \rceil$, and let $i' = \lfloor \frac{i}{b} \rfloor$ for all variables i. Unless j is a multiple of b, we change the constraint on h_j in equation (3.7) to the weaker constraint $h_j^{t_s} \ge h_{j-1}^{t_s}$. We similarly change the constraint on $A_j^{t_s}$. Furthermore, for $j \ge m'b$, we change the constraint on h_j in equation (3.7) to the weaker constraint $h_j = 1 - \beta$. We weaken constraint (3.6) by discarding the terms $1 - \epsilon + \beta$ (our lower bound for J_n) and e_j for j in s''s block $(s'b < j \le (s'+1)b)$. This changes the linear program of Lemma 3.7 to the following linear program: Minimize e_s subject to

$$(4.10)\quad m \ge \sum_{j=(s'+1)b}^{m} h_j^{t_s} + (s'+1)ba + \sum_{j=1}^{s'b} e_j$$

$$(4.11)\quad h_j^{t_s} \ge max(\alpha A_{j'b}^{t_s} - e_s, 1-\beta), \quad (s'+1)b \le j < m'b$$

$$(4.12)\quad h_j^{t_s} = 1-\beta, \quad m'b \le j < m$$

$$(4.13)\quad a \le A_{(s'+1)b}^{t_s}$$

$$(4.14)\quad A_{j'b}^{t_s} = \frac{1}{j'b}\sum_{k=0}^{j'b-1} h_k^{t_s}, \quad s'+2 \le j' < m'$$

Note that we have grouped all the machines into $x-1$ blocks of b machines with one last block of the tallest machines of size less than or equal to b (this follows from the assumption that $m > x(x-2)$). In particular, the linear programs for e_i and e_j are identical when $i' = j'$. Using this fact to simplify the equations, m is easily eliminated from equation (4.11). Lastly m is eliminated from equations (4.12) and (4.10) by pessimistically assuming that there are b machines in the last block (this is an upper bound). The linear program now simplifies to the following one: Minimize $e_{s'}$ subject to

$$\begin{aligned}
x &\ge \sum_{j'=s'+1}^{x-1} h_{j'}^{t_s} + (s'+1)a + \sum_{j'=1}^{s'-1} e_{j'} \\
h_{j'}^{t_s} &\ge max(\alpha A_{j'}^{t_s} - e_{s'}, 1-\beta) \\
&\qquad s'+1 \le j' < x-1 \\
h_{x-1}^{t_s} &= 1-\beta \\
A_{s'+1}^{t_s} &\ge a \\
A_{j'}^{t_s} &= \frac{1}{j'}\sum_{k=0}^{j'-1} h_k^{t_s}, \quad s'+2 \le j' < x-1
\end{aligned}$$

This linear program gives a valid bound for any $m > x(x-2)$. Therefore, we now have a technique to prove that for some α, ALG_α is α-competitive for $m > m_0$.

LEMMA 4.1. *When $\alpha \ge 1.945$ and $m > 13000$, ALG_α is α-competitive.*

Proof. We have written code that solves the linear program above, and for $\alpha = 1.945$ and $x = 115$, (and hence for $m \ge 115 \times 113 = 12995$), $A_s^{t_k}$ eventually falls below a and all elevating jobs that arrive after t_k are large. Again, by Corollary 3.2, the rest of the elevating jobs are large. By Lemma 3.3, the last job is large as well which means that there are $m+1$ large jobs. Therefore, in all cases $ALG_\alpha(\sigma) \le \alpha OPT(\sigma)$ and the lemma follows.

This completes the proof of our main result. We have actually shown that for $m > 20000^2$, ALG_α is α-competitive for $1.943 \le \alpha \le 2$ (by running the code once for $x = 20000$ and $\alpha = 1.943$). We expect that ALG_α is α-competitive for $1.943 \le \alpha \le 2$ for all m, but we lack the computing resources to verify 1.943 for all m up to 20000^2.

5 Lower Bounds

We now prove lower bounds on the performance of ALG_α for both small m and large m. We first prove for $m = 4$ and $m = 5$ that for no α is ALG_α better than $2 - \frac{1}{m}$-competitive. We then show that our analysis of ALG_α is almost tight by describing a sequence of jobs

such that $ALG_{1.9378}$ is not 1.9378-competitive on that sequence. Similar sequences can be constructed to prove that ALG_α is not 1.9378-competitive for $\alpha < 1.9378$, and these will appear in the journal version of this paper. Clearly, if $\alpha > 1.9378$, then for large enough m, ALG_α is no better than 1.9378-competitive on a long sequence of equal sized jobs.

5.1 Lower Bound for Small m

LEMMA 5.1. *For no α is ALG_α better than $(2-\frac{1}{m})$-competitive for $m = 4$ or $m = 5$.*

Proof. First consider the case for $m = 4$. For $\alpha \geq \frac{15}{7}$, ALG_α is clearly worse than 1.75-competitive on a long sequence of equal sized jobs. For $\alpha < \frac{15}{7}$, it is not hard to verify that ALG_α must act exactly like List on the sequence $1,1,1,1,2,2,2,2,4$ giving ALG_α a makespan of 7 while the optimal makespan is easily seen to be 4. Thus, for $m = 4$, $C_{ALG_\alpha} \geq 1.75$. The proof is almost identical for $m = 5$. For $\alpha \geq \frac{52}{27}$, ALG_α is clearly worse than 1.8-competitive on a long sequence of equal sized jobs. For $\alpha < \frac{52}{27}$, it is not hard to verify that ALG_α must act exactly like List on the sequence $1,1,1,1,1,2,2,2,2,2,5,5,5,5,5,10$ giving ALG_α a makespan of 18 while the optimal makespan is easily seen to be 10. Thus, for $m = 5$, $C_{ALG_\alpha} \geq 1.8$.

6 Lower Bound for Large m

6.1 Sketch

The lower bound job sequence σ, which demonstrates that $ALG_{1.9378}$ is not 1.9378-competitive for large m, is extracted from the upper bound proof, which shows that a sequence that is bad for ALG_α first causes it to create a flat schedule and then hits it with one final job of size 1. The method of constructing a flat schedule closely follows the kickstart.

The sequence σ makes the average machine height 1, as in the upper bound. The main part of σ (Phases 2 and 3) forces ALG_α to create a schedule with $(1-\lambda)m$ tall machines (with height $1-\epsilon$) and λm short machines (with height $a = 1/2-\epsilon$). The final part of σ (Phase 4) then forces ALG_α to follow the kickstart by repeatedly setting the height of the next job to be just large enough to prevent its going on any of the tall machines. The parameter λ is set so that the first jobs in Phase 4 have height $\frac{1}{2}$ so that two can be scheduled off-line on one machine. Finally, the last job of size 1 follows. The one complication is that we need to ensure that σ can be scheduled with a makespan of just 1. To do this, the first part of σ (Phase 1) consists of a large number of extremely small jobs which the off-line algorithm can use as filler.

The rest of this section is organized as follows. Sections 6.2 through 6.5 give details of phases 1 through 4. Section 6.6 shows how OPT can pack the jobs so that all machines have the same height. Throughout, we assume that m is large so that $O(1/m)$ terms can be ignored.

6.2 Phase 1

Phase 1 consists of a sequence of infinitesimal jobs which bring the average machine height to c, where $c = 0.2238$ (the reason for this value is given in Section 6.3).

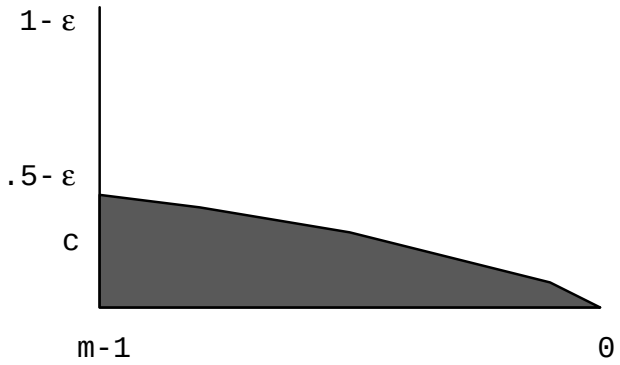

Figure 5: Schedule at end of Phase 1

The algorithm places the infinitesimal jobs so that the machine heights obey the relation $h_j = \alpha A_j$. This gives a recurrence for h_j, whose solution is

$$h_j = h_0 \binom{\alpha + j - 1}{j},$$

where the binomial coefficient $\binom{r}{k}$ is defined by $\binom{r}{k} = \frac{r(r-1)\ldots(r-k+1)}{k!}$ for real r and positive integer k, and $\binom{r}{k} = 1$ when $k = 0$. Then

$$A_j = \sum_{i<j} \frac{h_0}{j} \binom{\alpha + i - 1}{i} = \frac{h_0}{j} \binom{\alpha + j}{j}.$$

Since $A_m = c$, we have $h_0 = cm/\binom{\alpha+m}{m}$. Lastly, applying Stirling's approximation gives

$$h_j = c\alpha \left(\frac{j}{m}\right)^{\alpha-1} \left(1 + O\left(\frac{1}{j}\right)\right) \quad (6.15)$$

$$A_j = c \left(\frac{j}{m}\right)^{\alpha-1} \left(1 + O\left(\frac{1}{j}\right)\right) \quad (6.16)$$

6.3 Phase 2

Phase 2 consists of λm jobs which are placed on the current shortest machine and bring that machine's height to exactly $a = 1/2-\epsilon$. The value of c (the average machine height after Phase 1) is determined to ensure

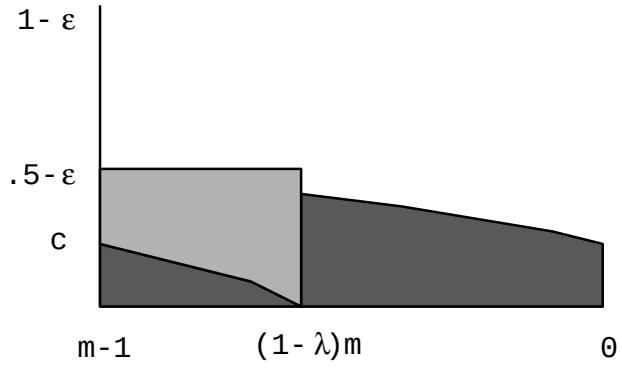

Figure 6: Schedule at end of Phase 2

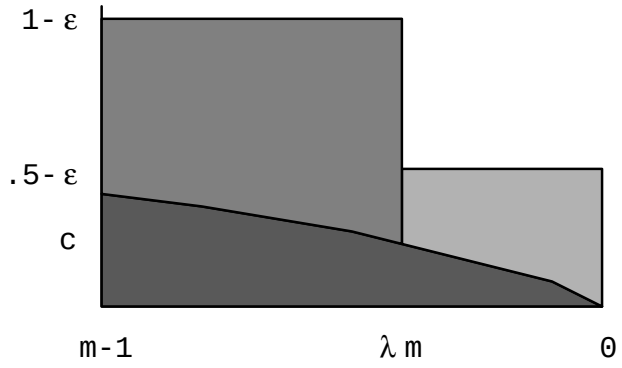

Figure 7: Schedule at end of Phase 3

that the algorithm does indeed place each job in Phase 2 on the machine with current smallest height.

The value of c is bounded by two constraints. First, a Phase 2 job must not be placed on a machine whose height is already a. Let A'_j and h'_j be the values of A_j and h_j respectively at the end of Phase 1. Since h'_j increases with j, the tightest constraint on c is when the last Phase 2 job is being inserted, and the constraint involves adding a job of size $a - h'_{\lambda m-1}$ and requiring that it be too tall to go on a height a machine:

$$\begin{aligned} & a + (a - h'_{\lambda m-1}) \\ & \quad > \quad \alpha A_{m-1} \\ & \quad = \quad \alpha \frac{a(\lambda m - 2) + mA'_m - (\lambda m - 1)A'_{\lambda m-1}}{m-1}, \end{aligned}$$

Substituting the approximations (6.15) and (6.16) above gives (for large m):

$$\begin{aligned} c \quad &< \quad \frac{a(2 - \alpha\lambda)}{\alpha(1 + \lambda^{\alpha-1} - \lambda^{\alpha})} \\ &\approx \quad 0.2238, \end{aligned}$$

using $\lambda = 0.392$ and $\epsilon = .0622$ (determined in Section 6.5). The second constraint on c is that a Phase 2 job must not be placed on a machine whose height is not yet a, but that is not the current smallest machine. Some straightforward calculations show that this is a weaker constraint on c. Hence we take $c = 0.2238$.

6.4 Phase 3

Phase 3 consists of $(1 - \lambda)m$ jobs which will be placed on the current shortest machine and bring that machine's height to exactly $1 - \epsilon$.

For this to work, we need to check that the algorithm will not place a Phase 3 job on any machine except the smallest at the time. The smallest job of Phase 3 has size $1 - \epsilon - c\alpha > 0.504$ (using $\epsilon = .0622$, determined below), so clearly no Phase 3 job will be placed on any nonminimal-height machine of height a or less. The average of the machine heights at any time in Phase 3 is at most $(1-\epsilon)(1-\lambda)+a\lambda \approx 0.742$, while adding any Phase 3 job to a machine of height $1 - \epsilon$ would raise it to height at least $1 - \epsilon + 0.504 = 1.442 > .742\alpha$. Hence the algorithm places each Phase 3 job on the smallest machine at the time.

6.5 Phase 4

Phase 4 consists of λm jobs where each job size is minimal subject to the constraint that the algorithm places the job on the smallest machine at the time. The parameter λ is chosen so that the first job of Phase 4 has size $1/2$, breaking the kickstart. Thus λ satisfies:

$$\begin{aligned} 1 - \epsilon + 1/2 \quad &> \quad \alpha A_{m-1} \\ &= \quad \alpha((1 - \epsilon)(1 - \lambda) + a\lambda), \end{aligned}$$

which gives λ as a function of ϵ. The constraint that the algorithm places the job on the smallest machine at the time gives a lower bound for J_j, which, after some calculations, is:

$$J_j > 1/2 + \frac{j(3/2 - \epsilon - \alpha a)}{m - j - 1},$$

so we set $J_j = 1/2 + \frac{j(3/2-\epsilon-\alpha a)}{m-j-1}$ (plus a tiny perturbation). Lastly, the fact that the set of all jobs has total size m determines a value for ϵ. Setting the sum of all the job sizes to be m and solving for ϵ gives $\epsilon = 0.0622$ and $\lambda = 0.392$.

6.6 Packing

To prove that $OPT = 1$, we need to show that the jobs can be assigned to machines so that each machine has height 1. This is done as follows: to make room for the job of size 1 (which gets its own machine), we

place the six smallest Phase 2 jobs on 2 machines, 3 per machine. We place the six smallest Phase 4 jobs on 3 machines, 2 per machine. Each of the remaining Phase 4 jobs gets its own machine. The remaining Phase 2 jobs are paired with Phase 3 jobs — the largest Phase 2 job with the smallest Phase 3 job, the second largest Phase 2 job with the second smallest Phase 3 job, and so on. Finally the infinitesimal Phase 1 jobs bring are used as filler to make each machine have height 1.

To check that this assignment does not make any machine too tall, observe the following facts:

- The smallest Phase 2 jobs have size $a - c\alpha\lambda^{\alpha-1} < 1/3$.
- The smallest Phase 4 jobs have size 1/2.
- The smallest Phase 3 job has size $1-\epsilon-c\alpha$, while the largest Phase 2 job has size $1/2-\epsilon$ for a total of less than 1. Furthermore, since the schedule after Phase 1 is convex, if the smallest Phase 3 job and the largest Phase 2 add to less than 1, then so do the second smallest Phase 3 job and the second largest Phase 2 job, and so on.

7 Extensions

We have been able to improve our algorithm slightly by melding it with the algorithm that compares to the average height of all machines.

DEFINITION 7.1. *The algorithm $ALG_{\alpha,\phi}$ works as follows: when a job J arrives, it is placed on the tallest machine k such that $h_k + J \leq \alpha((1-\phi)A_k + \phi A_{m+1})$.*

For a judicious choice of ϕ, $ALG_{\alpha,\phi}$ outperforms ALG_α. Unfortunately we can currently only prove a small improvement of about .001 in the competitive ratio for large m.

One unfortunate property of ALG_α is that its worst case performance ratio occurs on a set of identically sized jobs. One might desire an algorithm that performs well on such a job set, even at the cost of increasing the competitive ratio. We have shown such a tradeoff for ALG_α, namely that if $1.5 < \alpha < 2$, then there is an $\delta > 0$ such that ALG_α is $(2-\delta)$-competitive for large m. The details will be in the full paper. Clearly, for smaller α, ALG_α achieves a better makespan when all the jobs are the same size.

A related observation is that any algorithm which does well in the worst case cannot be optimal in the average case — it is easy to show that for any $\alpha < 2$ there is a δ such that any α-competitive algorithm, given a long sequence of identical jobs, must construct a schedule whose makespan exceeds $(1+\delta)OPT$.

This leaves three interesting open problems. First, to close the gap between the upper and lower bounds for $C(ALG_\alpha)$. Second, is ALG_α $(2-\delta)$-competitive for $\epsilon \geq .5$? And third, from the analysis of bad job sequences for ALG_α in Section 5, can one derive a better lower bound for all algorithms, or determine how an algorithm could improve on $C(ALG_\alpha)$?

8 Acknowledgements

We thank Alan Hu for contributing the example demonstrating that ALG_α cannot outperform List when $m = 4$.

References

[1] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24nd ACM Symp. on Theory of Computing*, pages 51–58, 1992.

[2] Y. Bartal, H. Karloff, and Y. Rabani. Private Communication.

[3] U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for particular problems. *Acta Cybernetica*, 9:107–119, 1989.

[4] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM J. Comput.*, 22:349–355, 1993.

[5] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[6] P. Narayanan and R. Chandrasekaran. Optimal on-line algorithms for scheduling. *Manuscript, University of Texas at Dallas*, 1991.

Appendix

We present here the full proof of Lemma 3.8.

Proof. Let $0 \leq \gamma \leq \epsilon$. Let a superscript on a variable denote the value of that variable with β equal to the superscript (e.g. b^γ is the value of b when $\beta = \gamma$). We show that if $e_1^\epsilon \cdots e_k^\epsilon > b^\epsilon$ and the linear program for e_{k+1}^ϵ is infeasible, then for some $j \leq k$, $e_1^\gamma \cdots e_j^\gamma > b^\gamma$ and the linear program for e_{j+1}^γ is infeasible.

Let $\delta = \epsilon - \gamma$. We have $b^\gamma = \frac{1}{2}\frac{1-\gamma}{1-\epsilon} = \frac{1}{2} + \frac{1}{2}\frac{\delta}{1-\epsilon}$ while $b^\epsilon = \frac{1}{2}$. Meanwhile, $a^\gamma = 1 - \gamma - \frac{1}{2}\frac{1-\gamma}{1-\epsilon} = \frac{1}{2} - \epsilon + \frac{1-2\epsilon}{2-2\epsilon}\delta$ while $a^\epsilon = \frac{1}{2} - \epsilon$.

This gives us the following two differences: $\Delta(b) = b^\gamma - b^\epsilon = \frac{1}{2-2\epsilon}\delta$ and $\Delta(a) = a^\gamma - a^\epsilon = \frac{1-2\epsilon}{2-2\epsilon}\delta$. We prove that for $1 \leq i \leq k$, if the linear program for e_i^γ is feasible, then $\Delta(e_i) = e_i^\gamma - e_i^\epsilon \geq \Delta(b)$, so e_i^γ is large if e_i^ϵ is large. If the linear program for e_i^γ is feasible for $1 \leq i \leq k$, then the fact that the linear program for e_{k+1}^ϵ is infeasible implies that the linear program for e_{k+1}^γ is also infeasible (since the constraints are all tighter in the latter program).

For the remainder of this proof, considering the linear program for elevating job e_s, $s \leq k$, arriving

at time t_s. We reserve the superscript on all variables (including h and A) to denote the value of β and omit the $0, t_s$.

Assume that $e_1^\gamma \cdots e_{s-1}^\gamma$ are indeed $\Delta(b)$ larger than their counterparts for $\beta = \epsilon$. We show that e_s^γ is as well by showing that if we set $e_s^\gamma = e_s^\epsilon + \Delta(b)$, and minimize h_i^γ subject to (3.7), then $h_i^\gamma \geq h_i^\epsilon$ for each of the tall machines M_i at this time. Let $\Delta(A_i) = A_i^\gamma - A_i^\epsilon$ and let $\Delta(h_i) = h_i^\gamma - h_i^\epsilon$ for each tall machine M_i. Using this notation, we want to prove that $\Delta(h_i) \geq 0$ for $i \geq s$. Proving this will be easy for i close to s because, in these cases, $h_i^\beta = 1 - \beta$ which implies $\Delta(h_i) = \delta$. The only problem we might have arises when h_i^ϵ finally rises above $1 - \epsilon$.

Let j be the minimal index where $\alpha A_j^\epsilon - e_s^\epsilon \geq 1 - \epsilon$. In this case, from Equation (3.7), we get $\Delta(h_j) \geq \alpha\Delta(A_j) - \Delta(b)$. If we can prove that $\Delta(h_j) \geq \Delta(A_j)$, then a simple induction proves that for $l \geq j$, $\Delta(A_l) \geq \Delta(A_j)$ which implies $\Delta(h_l) \geq \Delta(A_j) > 0$. Thus we need $(1 - \epsilon)\Delta(A_j) \geq \frac{1}{2-2\epsilon}\delta$ which means we need $\Delta(A_j) \geq \frac{1}{2(1-\epsilon)^2}\delta$. For $0 \leq \epsilon \leq .15$, this means we need $\Delta(A_j) \geq .7\delta$ while for $.15 \leq \epsilon \leq .2$, this means we need $\Delta(A_j) \geq .79\delta$.

If $j \geq 2s$, then $\Delta(A_j) \geq \frac{\Delta(a)s + \delta s}{2s} = \left(\frac{1-2\epsilon}{2-2\epsilon} + 1\right)\frac{\delta}{2} = \frac{3-4\epsilon}{4-4\epsilon}\delta$. For $\epsilon \leq .15$, this implies $\Delta(A_j) \geq .7\delta$ which, from the argument above, implies that for $l \geq j$, $\Delta(h_l) \geq 0$. Similarly, if $j \geq 4s$, $\Delta(A_j) \geq \frac{7-8\epsilon}{8-8\epsilon}$. For $.15 \leq \epsilon \leq .2$, this means $\Delta(A_j) \geq .84\delta \geq .79\delta$ which, from the argument above, implies that for $l \geq j$, $\Delta(h_l) \geq 0$. Thus we simply need to prove that for $0 \leq \epsilon \leq .15$ that h_i^ϵ does not rise above $1 - \epsilon$ for $i < 2s$ and that for $.15 \leq \epsilon \leq .2$, that h_i^ϵ does not rise above $1 - \epsilon$ for $i < 4s$.

For the remainder of this proof, $\beta = \epsilon$. Let j be the first point where $\alpha A_j - e_s \geq 1 - \epsilon$. Clearly, this is the first point where h_j rises above $1 - \epsilon$. Substituting $\frac{s(\frac{1}{2}-\epsilon)+(j-s)(1-\epsilon)}{j}$ for A_j and simplifying leads to $j \geq \frac{2-\epsilon}{1-4\epsilon+2\epsilon^2}s$. For $\epsilon \geq 0$, $\frac{2-\epsilon}{1-4\epsilon+2\epsilon^2} \geq 2$ while for $.2 \geq \epsilon \geq .15$, $\frac{2-\epsilon}{1-4\epsilon+2\epsilon^2} \geq 4$. This completes the proof of the lemma.