Busy-Time Scheduling on a Bounded Number of Machines

Frederic Koehler[1] and Samir Khuller[2]

[1] Dept. of Mathematics, MIT, Cambridge MA 02139 USA
fkoehler@mit.edu
[2] Dept. of Computer Science, Univ. of Maryland, College Park MD 20742 USA
samir@cs.umd.edu

Abstract In this paper we consider a basic scheduling problem called the busy time scheduling problem - given n jobs, with release times r_j, deadlines d_j and processing times p_j, and m machines, where each machine can run up to g jobs concurrently, our goal is to find a schedule to minimize the sum of the "on" times for the machines. We develop the first correct constant factor online competitive algorithm for the case when g is unbounded, and give a $O(\log P)$ approximation for general g, where P is the ratio of maximum to minimum processing time. When g is bounded, all prior busy time approximation algorithms use an unbounded number of machines; note it is NP-hard just to test feasibility on fixed m machines. For this problem we give both offline and online (requiring "lookahead") algorithms, which are $O(1)$ competitive in busy time and $O(\log P)$ competitive in number of machines used.

1 Introduction

Scheduling jobs on multiple parallel machines has received extensive attention in the computer science and operations research communities for decades (see reference work [3]). For the most part, these studies have focused primarily on job-related metrics such as minimum makespan, total completion time, flow time, tardiness and maximum throughput. Our work is part of a line of recent results working towards a different goal: *energy efficiency*, in particular aiming to minimize the total time that a machine must be turned on, its *busy time* [4,14,9,13,17,5]. Equivalently, we seek to maximize the *average load* of machines while they are powered on, assuming we are free to turn machines off when they are idle. Note in this context we are concerned with *multi-processing* machines, as for machines which process only one job at a time the load is either 0 or 1 always. This measure has been studied in an effort to understand energy-related problems in cloud computing contexts; see e.g. [13,5,4] . The busy time metric also has connections to several key problems in optical network design, for example in minimizing the fiber costs of Optical Add Drop Multiplexers (OADMs) [9], and the application of busy time models to optical network design has been extensively outlined in the literature [9,10,11,20,1].

Formally the problem is defined as follows: we are given a set of n jobs, and job j has a release time of r_j, a deadline d_j and a processing time of p_j (it is assumed $r_j + p_j \leq d_j$) and a collection of m multiprocessor machines with g processors each. The significance of processors sharing a machine is that they share busy time: the machine is on if a single processor on the machine is active. Each job j is assigned to the time window $[s_j, s_j + p_j)$ on some machine m_j. The assignment must satisfy the following constraints:

1. Start times respect job release times and deadlines, i.e., $[s_j, s_j + p_j) \subseteq [r_j, d_j)$.
2. At most g jobs are running at any time on any given machine. Formally, at any time t and on any machine m, $|\{j | t \in [s_j, s_j + p_j), m_j = m\}| \leq g$.

The busy time of a machine is the duration for which the machine is processing any non-zero number of jobs. The objective is to minimize the total sum of busy times of all the machines. Formally, the objective function is

$$\sum_{i=0}^{\infty} \mu \left(\bigcup_{j:m_j=i} [s_j, s_j + p_j) \right)$$

where μ measures the geometric length of a union of disjoint half intervals by summing their individual lengths; e.g. $\mu([1,2) \cup [3,4) \cup [3,5)) = 3$ i.e. μ is Lebesgue measure. *Note that this objective is constant if $g = 1$.*

All previous algorithms (described below) for busy time are forced to make the assumption that $m = \infty$, because the number of machines required by the schedules they generate can be as large as $\Omega(n)$, i.e. worst-possible. Our primary interest in this paper is in improving on this front. Thus our primary interest will really be in the *simultaneous optimization* problem of generating schedules whose performance is bounded in two objectives simultaneously: both the busy time and the number of machines required by the schedule. The best known approximation algorithms for each of these objectives separately is 3 [5] and $O(\sqrt{\log n / \log \log n})$ [6]. We conjecture that there exist schedules which achieve a $O(1)$ approximation in *both* objectives. However, as it stands the $O(1)$ machine minimization problem by itself remains a major open problem in combinatorial optimization, so such a result is out of reach for now. The main result of our paper will show that we can at least construct such a schedule under the assumption that $\log P$ is bounded by a constant, where $P = \max_{i,j} p_j / p_i$.

1.1 Related Work

Winkler and Zhang [20] first studied the interval job case of busy time scheduling, i.e. when $p_j = d_j - r_j$, and showed that even the special case when $g = 2$ is NP-hard. Their work was motivated by a problem in optical communication and assigning routing requests to wavelengths. Assuming that the number of machines available is unbounded, Alicherry and Bhatia [1], and independently Kumar and Rudra [15], developed approximation algorithms with an approximation factor of 2 for the case of interval jobs. Being unaware of prior work

on this problem, subsequently, Flammini et al [9] developed a very simple 4 approximation via a greedy algorithm for the interval job case.

The first constant factor approximation for the general problem, albeit on an unbounded number of machines, was was given by Khandekar et al [13]. They first design a dynamic programming based algorithm for the case when $g = \infty$. This schedule is then used to fix the starting times of the jobs, and the resulting instance of interval jobs is scheduled by the greedy algorithm of Flammini et al [9]. Despite the "restriction" mapping, the approximation factor of 4 is unchanged. Since better approximation algorithms for the interval job case were available, it is natural to attempt to use those instead. Sadly, the restriction mapping can actually increase the cost of an optimal solution by a factor of 2, and so even if we use these algorithms we do not get a better bound than 4 (see [5] for a tight example). Chang et al [5] developed a 3-approximation algorithm by giving a new interval packing algorithm. We conjecture that the correct answer for this is 2, matching the interval job case.

Unfortunately, the number of machines used by all of these algorithms may be as large as $\Omega(n)$, even in the case when all jobs are equal length and released at time $r_j = 0$. This is because the $g = \infty$ reduction chooses start times oblivious to the true value of g. One may hope to resolve this problem from the other direction, by adapting an algorithm for minimizing the number of machines used. It is not difficult to get a $O(\log n)$ approximation algorithm for this problem via randomized LP rounding. The best known result is a $O(\sqrt{\log n / \log \log n})$ approximation algorithm by Chuzhoy et al [6] which uses a sophisticated recursive LP relaxation to the problem. Unfortunately, it appears to us quite difficult to adapt these LP rounding methods to account for the cost of the nonlinear busy time objective.

When $g < \infty$, very strong lower bounds for online minimization of busy time were given by Shalom et al [18]. They show that when $g < \infty$, no online algorithm can be better than g competitive algorithm against an online adaptive adversary. It should be noted that their general online model is harder than the one we consider; they have no notion of time, so in the online scenario they envision the algorithm must be able to deal with jobs arriving in arbitrary order. However, their proof of the lower bound does not need this additional power: it releases jobs in left-to-right order.

Some recent work [8,12] claims a 2-competitive online algorithm when $g = \infty$, but it is incorrect; see Fig. 1.

1.2 Our Contributions

We divide the results into sections depending on the flexibility the algorithm has with m, the number of machines. We begin with the "classic" busy time model, where $m = \infty$.

- Our first result is an online 5-competitive algorithm for the busytime problem when machine capacity is unbounded $g = \infty$. In addition, we show that against an adaptive online adversary there is no online algorithm with competitive ratio less than ϕ.

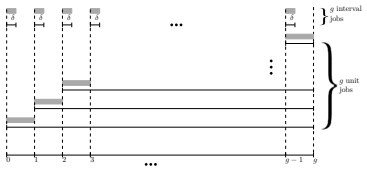


Figure 1: Counter-example to online algorithm of [8]. The optimal solution delays all the flexible unit jobs to the end and gets a busy time cost of $1 + g\delta$ rather than g. Setting $\delta = \frac{1}{g}$ gives the gap. The figure shows the schedule produced by the online algorithm with a cost of g.

– The previous result is extended to the general busy time problem with $g < \infty$, and we get a competitive ratio of $O(\log P)$. No online algorithm for this problem was previously known. In the online setting with lookahead of $2p_{max}$ we can give a 12-competitive algorithm.

We then present our main results, concerned with simultaneous optimization of busytime and number of machines used:

– We present a constant-factor approximation algorithm for the busy time problem with fixed number of machines m, given the assumption of identical length jobs $p_j = p$.
– We give the first approximation algorithm for busy time scheduling with a non-trivial bound on the number of machines used. More precisely, for the simultaneous optimization problem we give a schedule which is $3 + \epsilon$-competitive on busy time and $O(\frac{\log P}{\log(1+\epsilon)})$ competitive on machine usage for $\epsilon < 1$.
– We give an online algorithm with $O(p_{max})$ lookahead in time, which remains $O(1)$-competitive for busy time and $O(\log P)$ competitive on machine usage.
– We also give *tradeoff lower bounds* which show the limits on the simultaneous optimizability of these objectives; if we optimize solely for one objective (e.g. machine usage), we may lose a factor of $\Omega(g)$ in the other (e.g. busy time).

1.3 Preliminaries

We recall the following fundamental scheduling lemma. The *interval graph* of a collection of half-intervals $\{[\alpha_i, \beta_i)\}_{i=1}^n$ is the graph with vertices the half-intervals, and an edge between two half-intervals I_1 and I_2 iff $I_1 \cap I_2 \neq \emptyset$. The interval graph is perfect, i.e.:

Proposition 1 *Given a collection of half-open intervals $\{[\alpha_i, \beta_i)\}_{i=1}^n$ there exists a k-coloring of the corresponding interval graph iff for all $t \in \mathbb{R}$,*

$$|\{i : [\alpha_i, \beta_i) \ni t\}| \leq k. \tag{1}$$

Proposition 2 *The following are lower bounds on the optimum busy time:*

1. *The optimal busy time for the same input instance with* $g = \infty$.
2. *The load bound* $(1/g)\sum_{j=1}^{n} p_j$.

We say a job is *available* at time t if $r_j \leq t$. It is often useful to refer to the latest start time $u_j = d_j - p$ of a job. An *interval job* is one with no choice in start time, i.e. j is an interval job when $d_j - r_j = p_j$. We define an algorithm to be a (r_1, r_2)-approximation if it generates a schedule using at most $r_1 m_{opt}$ machines and $r_2 \text{busy}_{OPT}$ busy time, where m_{opt} is the smallest number of machines for which a feasible schedule exists, and busy_{OPT} (or just OPT) is the minimum busy time on an unbounded number of machines.

2 Online Busy Time Scheduling with an Unbounded Capacity

2.1 The $g = \infty$ case, Upper Bound

We give a 5-competitive deterministic online algorithm for busy time scheduling when $g = \infty$. (We will also show this analysis is tight.) In this setting we may assign all jobs to a single machine so we assume w.l.o.g. $m = 1$. Informally, the algorithm is quite simple: everytime we hit a latest starting time u_j of an unscheduled job j, we activate the machine from time u_j to time $u_j + 2p_j$ and run all the jobs that fit in this window. To analyze this, we can pick an *arbitrary* optimal schedule, decompose its busy time into connected components, and then bound the cost of our schedule by charging the cost of running jobs to the connected components containing them.

In this section we will let T denote the *active time* of our machine; all jobs are processed during this active time, i.e. $\bigcup_j [s_j, s_j + p_j) \subset T$. We also maintain a set P of *primary jobs* but this is only for the purposes of the analysis.

Algorithm Doubler:

1. Let $P = \emptyset$. Let $T = \emptyset$.
2. For $t = 0$ to d_{max}:
   (a) Let U be the set of unscheduled, available jobs.
   (b) Run every unscheduled job j s.t. $[t, t + p_j) \subset T$; j is now removed from U.
   (c) If $t = u_j$ for some $j \in U$, then pick such a j with p_j maximal and set $T = T \cup [t, t + 2p_j)$ (i.e. activate the machine from time t to $t + 2p_j$). Let $P = P \cup \{j\}$.
   (d) Once again, run every unscheduled job j s.t. $[t, t + p_j) \subset T$; j is removed from U.

Suppose the algorithm fails to schedule a job j. Then at time u_j the job was available but was not scheduled; impossible because steps 2(c) ensures that $T \supset [u_j, u_j + p_j)$ and so step 2(d) would necessarily schedule it. Thus the algorithm schedules all jobs and, because we may trivially verify it respects r_j, d_j constraints, produces a valid schedule. Henceforth s_j refers to the start times chosen by algorithm Doubler; the following proposition is immediate.

Proposition 3 *Here T and P are as above upon completion of the algorithm. For every $j \in P$, $s_j = u_j$; also $T = \bigcup_{j \in P} [s_j, s_j + 2p_j)$.*

Theorem 1. *Algorithm Doubler is 5-competitive.*

Proof. Fix an input instance (r_j, d_j, p_j) and an optimal offline schedule OPT with start times s_j^*. Let $T^* = \bigcup_j [s_j^*, s_j^* + p_j)$ so $\mu(T^*)$ is the busy time cost of OPT. Let P be the set of primary jobs. Let $P_1 \subset P$ consist of those jobs j in P with $[s_j, s_j + 2p_j) \subset T^*$ and $P_2 = P \setminus P_1$. By the Proposition,

$$\begin{aligned} \mu(T) &= \mu\left(\bigcup_{j \in P} [s_j, s_j + 2p_j)\right) \\ &\leq \mu\left(\bigcup_{j \in P_1} [s_j, s_j + 2p_j)\right) + \mu\left(\bigcup_{j \in P_2} [s_j, s_j + 2p_j)\right) \\ &\leq \mu(T^*) + \sum_{j \in P_2} 2p_j. \end{aligned} \tag{2}$$

It remains to bound the cost incurred by jobs in P_2. Decompose T^* into connected components $\{\mathcal{C}^i\}_{i=1}^k$ so $T^* = \mathcal{C}^1 \cup \cdots \cup \mathcal{C}^k$. The endpoints of $\mathcal{C}^i$ are $\inf \mathcal{C}^i$ and $\sup \mathcal{C}^i$. Let $J(\mathcal{C}^i)$ be the set of jobs j with $[s_j^*, s_j^* + p_j) \subset \mathcal{C}^i$. OPT schedules all jobs so $\bigcup_i J(\mathcal{C}^i)$ is the set of all jobs, thus $\sum_{j \in P_2} 2p_j = \sum_{i=1}^k \sum_{j \in P_2 \cap J(\mathcal{C}_i)} 2p_j$. We now claim that

$$\sum_{j \in P_2 \cap J(\mathcal{C}_i)} p_j \leq 2\mu(\mathcal{C}^i). \tag{3}$$

To show the claim, first we index so $\{e_j^i\}_{j=1}^{k_i'} = P_2 \cap J(C_i)$, where $k_i' = |P_2 \cap J(C_i)|$, and $(s(e_j^i))_{j=1}^{k_i'}$ is a monotonically increasing sequence.

Observation: $r_{e_j^i} \leq s(e_1^i)$ for all j. Suppose for contradiction that $r_{e_j^i} > s(e_1^i)$ for some j. We know $[s^*(e_j^i), s^*(e_j^i) + p_{e_j^i}) \subset \mathcal{C}^i$, hence $r_{e_j^i} + p_{e_j^i} \leq \sup \mathcal{C}^i$. Because $[s(e_1^i), s(e_1^i) + 2p_1) \not\subset \mathcal{C}^i$ we know that $s(e_1^i) + 2p_1 \geq \sup \mathcal{C}_i \geq r_{e_j^i} + p_{e_j^i}$. Thus $[r_{e_j^i}, r_{e_j^i} + p_{e_j^i}] \subset [s(e_1^i), s(e_1^i) + 2p_1) \subset T$. We see then that at time $r_{e_j^i}$, step 2 (b) the algorithm must have scheduled job e_j^i. Thus $e_j^i \notin P \supset P_2 \cap J(C_i)$, which contradicts the definition of e_j^i. By contradiction $r_{e_j^i} \leq s(e_1^i)$ for all j.

Now it follows that $p_{e_j^i} > 2p_{e_{j-1}^i}$ (for $j \geq 2$): suppose otherwise, then because we know e_j^i was available at step 2 (c) at $t = s(e_{j-1}^i) \geq s(e_1^i) \geq r_{e_j^i}$, job e_j^i must have been scheduled at t with e_{j-1}^i and cannot have been added to P. By contradiction, $p_{e_j^i} > 2p_{e_{j-1}^i}$ hence by induction $p_{e_{k_i'}^i} > 2^{k_i' - j} p_{e_j^i}$. Now (3) follows:

$$\sum_{j=1}^{k_i'} p_{e_j^i} \leq \sum_{j=1}^{k_i'} 2^{j - k_i'} p_{e_{k_i'}^i} < p_{e_{k_i'}^i} \sum_{j'=0}^{\infty} 2^{-j'} = 2p_{e_{k_i'}^i} \leq 2\mu(\mathcal{C}^i).$$

Thus $\sum_{j \in P_2} 2p_j \leq \sum_{i=1}^{k} 4\mu(\mathcal{C}^i) = 4\mu(T^*)$. Combining this with (2) proves the theorem.

Obviously we could have defined the above algorithm replacing 2 with any $\alpha > 1$, however $\alpha = 2$ minimizes $\alpha + \sum_{i=0}^{\infty} \alpha^{-i}$ and is thus optimal.

This analysis is tight. Fix $N > 0$ and let $\epsilon = 1/N$. Consider an instance where we release $2^N N$ rigid jobs of length ϵ with availabity constraints $[0, \epsilon), [\epsilon, 2\epsilon), \ldots, [(2^N N - 1)\epsilon, 2^N)$, and also at the beginning of time release flexible jobs of length $1+\epsilon, 2+(2+\epsilon)^1\epsilon, 4+(2+\epsilon)^2\epsilon, \ldots, 2^N+(2+\epsilon)^N\epsilon$ with deadlines $2^N+(1+\epsilon), 2^N+2(1+\epsilon)+2(2+(2+\epsilon)^1\epsilon)$, and so on so that each latest start time is twice the previous processing time plus the previous latest starting time. For ϵ sufficiently small, every job will become primary and hence, taking $\epsilon \to 0$ the machine will be activated for total time $2^N + 2(2^N + 2^N) = 5 * 2^N$ whereas, by starting all of the flexible jobs at time 0 we would need only 2^N busy time.

2.2 $g = \infty$, Online Lower Bounds

Proposition 4 *No online algorithm (without lookahead) against an online adaptive adversary has competitive ratio better than* $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Proof. Let $0 < \alpha < 1$ be a constant to be optimized later. Fix $1 > \epsilon > 0$ such that $\alpha = \epsilon k$ where $k \in \mathbb{Z}$. Here is the strategy for the adversary:

1. Release job A of length 1 available in $[0, 3)$.
2. Until job A is started, at each $t = n\epsilon$ for $n < k \in \mathbb{Z}$ release a single job of length ϵ available in $[t, t+\epsilon)$. (The ϵ jobs are interval jobs.)
3. If job A was started at $t = n\epsilon$, release a final job of length 1 available in $[2, 3)$.
4. Otherwise if job A is still not started at time $(k-1)\epsilon$, release no more jobs.

In the case corresponding to step (3), the online schedule has busy time $n\epsilon+1+1$ whereas the optimal offline schedule, which runs job A at time 2, has busy time $(n+1)\epsilon+1$. The ratio is thus $\frac{n\epsilon+2}{(n+1)\epsilon+1} \geq \frac{\alpha-\epsilon+2}{\alpha+1}$ because $f(x) = \frac{x-\epsilon+2}{x+1}$ is monotonically decreasing for $x > 0$. In the case corresponding to step (4), the online schedule has busy time at least $(k-1)\epsilon+1 = \alpha-\epsilon+1$ whereas the offline schedule has busy time 1. Thus the competitive ratio is at least $\min\left\{\frac{\alpha-\epsilon-2}{\alpha+1}, \alpha-\epsilon+1\right\}$ and we may take the limit as $\epsilon \to 0$. The positive solution to $\frac{\alpha-2}{\alpha+1} = \alpha+1$ is at $\alpha = \frac{\sqrt{5}-1}{2}$, and thus we get a lower bound of $\varphi = \frac{1+\sqrt{5}}{2}$.

A similar proof also gives a weaker lower bound when the algorithm is granted lookahead of $O(p_{max})$. Let $0 < \beta < 1$. Release job A with a very large availability span, and simultaneously release an interval job of length β, i.e. a job with $r_j = 0, p_j = \beta, d_j = \beta$. Without loss of generality the online algorithm either schedules job A at time 0 or chooses not to schedule job A until after time β. In the former case, release a job of length 1 at the very end of job A's availability window; in the latter case, release no more jobs. The lower bound on the competitive ratio now $\min\{\frac{1+\beta}{1}, \frac{2}{1+\beta}\}$, optimized at $\beta = \sqrt{2}-1$, giving a ratio of $\sqrt{2}$.

Proposition 5 *An algorithm with lookahead a function of p_{max} has competitive ratio at least $\sqrt{2} \approx 1.414$.*

2.3 General Case, $g < \infty$

Combining with the bucketing algorithm given by Shalom et al [18] this gives a $O(\log \frac{p_{max}}{p_{min}})$-competitive online algorithm for busy time scheduling. More precisely, because the cost of their algorithm is bounded by 4 times the weight of the input jobs, and 1 times the $g = \infty$ lower bound, the approximation is $9 \log \frac{p_{max}}{p_{min}}$.

Running Algorithm Doubler offline and combining with the 3-approximation of Chang et al [5] gives a fast 7-approximation to the optimal busy time schedule. This is because the Greedy Tracking algorithm [5] takes as input a $g = \infty$ schedule using busytime T and outputs a schedule with cost at most $T + 2w(J)/g \leq T + 2OPT$ where $w(J)$ denotes the total processing time of all jobs. Since $T \leq 5OPT$ using our algorithm, the cost is bounded by $7OPT$.

If we are willing to grant the online algorithm a lookahead of $2p_{max}$ then we can get a constant factor online algorithm. We use our $g = \infty$ online algorithm to determine the time placement of jobs; this algorithm requires no lookahead so we now know the start time of jobs $2p_{max}$ ahead of the current time. We now run the offline machine-assignment algorithm in windows of the form $[kp_{max}, (k+2)p_{max})$ for $k \in \mathbb{N}$. We can bound the cost of even k by $5OPT + 2w(J_0)/g$ where $w(J_0)$ is the total processing time of jobs run in windows with even k; adding the matching term for odd k shows that this gives a fast $2 * 5 + 2 = 12$ approximation.

3 Offline Algorithm for Equal length jobs, Bounded Number of Machines

Although it is impossible in the general case (see Lower Bounds, Section 6), in the case of $p_j = p$ we are able to compute a schedule which is $(1, O(1))$-approximate, i.e. with the optimal number of machines and $O(1)$ busy time vs. the busy time optimum. A lower bound shows that a $(O(1), 1)$-approximation is impossible to achieve, even in this scenario. Our algorithm is two-step: it starts with a feasible schedule, and then uses a "pushing scanline" to push jobs together and reduce the busytime cost. For space reasons the analysis has been moved to the appendix; below is the algorithm.

Algorithm Compact

1. Compute a feasible schedule S with the minimum number of machines by binary search ($0 \leq m_{opt} \leq n$), using a feasibility algorithm for the problem with mg identical machines, each with one processor. The best known algorithm is the $O(n^2)$ feasability algorithm of [16].
2. Ensure that S is *left-shifted*. Let s_j^0 denote the start time of job j in S, and let $s_j := s_j^0$. Let $K := \emptyset$ and $P := \emptyset$.
3. For t from r_{min} to d_{max}: (*main loop*)
   (a) For every unscheduled job j, let $s_j := \max\{s_j^0, t\}$. Let U be the set of unscheduled jobs.

(b) If $|\{j \in U : s_j \in [t, t+2p]\}| \geq mg$, run each job j in this set at time s_j. Let $K := K \cup \{[t, t+3p)\}$. We say these jobs were run in a *cluster*. Return to the main loop at $t := t + 2p$.
(c) Otherwise if $t = u_j$ for some unscheduled job j, run each job in the set $\{j \in U : s_j \in [t, t+p]\}$ at its start time s_j. Return to the main loop at $t := t + p$. Let $P := P \cup \{j\}$.

In step 3 it is necessary to consider only $t \in \{u_j, s_j - 2p\}$, so we can run this step in linearithmic time.

4 Offline Algorithm for Bounded Number of Machines

In this section we will use the fact that scheduling jobs on a minimum number of machines with integer release times and deadlines and with $p = p_j = 1$ is trivial offline. For a fixed m, it is well-known that an EDF (earliest-deadline first) schedule, i.e. one given by running at each time up to m of the jobs with earliest deadlines, gives a feasible schedule iff the input instance is feasible. Computing the minimum m can be done by binary search in $\log n$ steps.

We would like to describe some intuition before launching into the formal analysis. As before, we use something like a "pushing scanline" approach, starting from a left-shifted schedule and starting a group of jobs whenever a large number have been pushed together. To make this approach have bounded busy time usage, we need to bucket jobs by similar lengths and use the approach on a per-bucket basis, but this alone cannot attain our desired performance ratio, because we may need, for busy time purposes, to group some long jobs with some short jobs. Therefore, in each bucket, when a job does not nicely group together with other jobs of the same length, we temporarily drop it. A second "clean-up pass" (step 3 below) runs the remaining jobs using an algorithm which has good busy-time performance but a priori unbounded machine usage. By arguing that we drop few jobs with overlapping availability times from each bucket, it is then possible to bound the machine usage (see proof of Theorem 4).

We now present a $(O(\log p_{max}/p_{min}), O(1))$-approximation algorithm for the general problem. Fix a constant $\alpha > 1$ to be optimized later.

1. Bucket jobs by processing time increasing exponentially by α, so the buckets contain jobs of processing time in the intervals $[p_{min}, \alpha p_{min})$, $[\alpha p_{min}, \alpha^2 p_{min}), \ldots,$ $[\alpha^{q-1} p_{min}, \alpha^q p_{min}]$ where $q = \left\lceil \log_\alpha \frac{p_{max}}{p_{min}} \right\rceil$.
2. For each bucket B_i
   (a) Let p be the supremum of the processing times of jobs in this bucket. We round job availability constraints down to multiples of p, so $r'_j = p\lfloor r_j/p \rfloor, u'_j = p\lfloor u_j/p \rfloor$, and $p'_j = p$. This is a unit job scheduling problem after we rescale by a factor of p.
   (b) We generate a left-shifted feasible schedule (referred to as the *initial schedule*) for the rounded (r'_j, d'_j, p'_j) instance using the minimum number of machines m. Let s^0_j be the start time of job j in this schedule.

(c) Execute Algorithm RunHeavy.
(d) Let U_i' denote the set of jobs unscheduled in B_i after running Algorithm RunHeavy.

3. Now let U'' be the union of the U_i' for all buckets, and schedule the jobs in U'' by the 3-approximation of Chang et al [5] upon a new set of machines (we will bound the number needed).

Algorithm RunHeavy

1. Let U initially be the set of all jobs in the bucket. Split machines into groups M_1 and M_0; we will require at most m machines in each group (see Proposition 14 below).
2. For $t = kp$ from r'_{min} to u'_{max}:
   (a) Let $J_t = \{j \in U : s_j^0 = t\}$. Let $k_1 = \lfloor |J_t|/g \rfloor$ and run k_1 groups of g jobs from this set on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s_j^0, r_j)$. Remove these jobs from U.
   (b) Let $J_t' = \{j \in U : s_j^0 \leq t \leq u_j'\}$. Let $k_2 = \lfloor |J_t'|/g \rfloor$ jobs, and run k_2 groups of g jobs from this set on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s_j^0, r_j)$. Remove these jobs from U.

Note in the loop in RunHeavy, we only need to do something when $t = s_j^0$ for some job j so the loop is really over polynomially many t. The analysis of this algorithm is in the Appendix; we show the resulting schedules requires at most $q(2\lceil\alpha\rceil m_{opt} + 8)$ machines and $(2\alpha + 1)OPT$ busytime.

5 Online Algorithm for Bounded Number of Machines

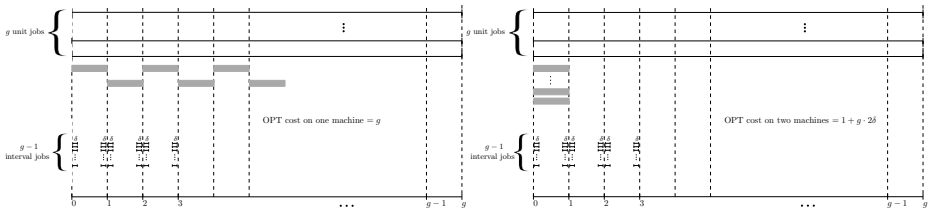
Since the formal details in this section are quite long, we give a brief summary of the main idea. In order to get an online algorithm, we still use the approach of the previous section, but interweave an agressive variant of Algorithm Doubler in order to pick start times for the "leftover" jobs which fit poorly into their buckets. The full result is included in the Appendix. Also, in the previous section we used that the $(r_j, d_j, p = p_j = 1)$ problem was exactly solvable offline. In this section, we instead rely upon the online e-competitive algorithm of [2] for this same problem, and then use our $g = \infty$ algorithm in order to schedule the jobs in U'' online with bounded performance.

6 Simultaneous Optimization of Busy Time and Machine Usage

6.1 Lower Bounds

Proposition 6 *For any input g, there exist input instances (with g processors per machine) where every machine-optimal schedule uses $(g-\epsilon)busy_{opt}$ busy time for ϵ arbitrary small.*

Proof. Fix $1 > \delta > 0$. Release g jobs of length 1 at time 0 with availability windows $[0, g)$. For $k = 0$ to $g - 1$, release $g - 1$ jobs of length δ with availability windows $[k, k + \delta)$, and $g - 1$ jobs of length δ with availability windows $[k + 1 - \delta, k + 1)$. The machine-optimal schedule runs all jobs on one machine, but due to the presence of the δ-jobs cannot overlap the execution of the long jobs, and thus has busy time cost g (see Fig. 2a). The busy time optimal schedule runs the δ jobs on a separate machine and runs all of the long jobs in parallel, giving a busy time cost $1 + 2g\delta$. Thus the ratio is $\frac{g}{1+2g\delta}$ and taking δ sufficiently small gives the desired result.



(a) On one machine the busytime is g. (b) On two machines the busytime is $1 + 2g\delta$.

Figure 2: Illustrations of trade-off lower bounds.

Proposition 7 *For any g, there exist input instances where every busy time optimal schedule uses gm_{opt} machines, even with the restriction $p_j = p$.*

Proof. We set $p = p_j = 1$ for all jobs. For $k = 0$ to $g - 1$, we release an interval job with availability window $[k/g, k/g+1)$, and we release $g(g-1)$ *unconstrained* jobs with availability windows $[0, 2g^2)$.

There exists a busy time optimal schedule using g machines, which runs $g-1$ unconstrained jobs along with a single interval job together on a machine. Here the busy time cost equals the load bound exactly. There exists a feasible schedule using only 1 machine: for $k = 0$ to $g - 1$, on processor k of the machine it runs first the interval job followed by $g - 1$ unconstrained jobs, end-to-end. Thus $m_{opt} = 1$.

Now consider any schedule using fewer than g machines. By the pigeonhole principle, it must run two interval jobs on a single machine M. Let these jobs start at k_1/g and k_2/g respectively with $k_1 < k_2$; then the processor running the job at k_2/g must be idle in $[0, k_2/g) \supset [k_1/g, k_2/g)$. Since the load is positive but below g in this interval, the busy time exceeds the busy time lower bound, and so is greater than the cost of the busy time optimal schedule described earlier.

Acknowledgements: We are grateful to Chunxing Yin for extremely useful discussions.

References

1. Mansoor Alicherry and Randeep Bhatia. Line system design and a generalized coloring problem. In *ESA*, pages 19–30, 2003.
2. Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1):3:1–3:39, March 2007.
3. Peter Brucker. *Scheduling algorithms.* Springer, 2007.
4. Jessica Chang, Harold Gabow, and Samir Khuller. A model for minimizing active processor time. In *ESA*, pages 289–300, 2012.
5. Jessica Chang, Samir Khuller, and Koyel Mukherjee. Lp rounding and combinatorial algorithms for minimizing active and busy time. In Guy E. Blelloch and Peter Sanders, editors, *SPAA*, pages 118–127. ACM, 2014.
6. Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Seffi Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS*, pages 81–90. IEEE, 2004.
7. Nikhil R. Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online algorithms for machine minimization. *CoRR*, abs/1403.0486, 2014.
8. Xiaolin Fang, Hong Gao, Jianzhong Li, and Yingshu Li. Application-aware data collection in wireless sensor networks. In *Proceedings of INFOCOM*, 2013.
9. Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Hadas Shachnai, Mordechai Shalom, Tami Tamir, and Shmuel Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *IPDPS*, pages 1–12, 2009.
10. Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *Proceedings of Euro-Par*, pages 920–929, 2008.
11. Michele Flammini, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem. In *ISAAC*, pages 915–924, 2005.
12. Chi Kit Ken Fong, Minming Li, Shi Li, Sheung-Hung Poon, Weiwei Wu, and Yingchao Zhao. Scheduling tasks to minimize active time on a processor with unlimited capacity. In *MAPSP*, 2015.
13. Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *FSTTCS*, pages 169 – 180, 2010.
14. Frederic Koehler and Samir Khuller. Optimal batch schedules for parallel machines. In *WADS)*, pages 475–486, 2013.
15. Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *FSTTCS*, pages 152–163, 2005.
16. Alejandro López-Ortiz and Claude-Guy Quimper. A fast algorithm for multimachine scheduling problems with jobs of equal processing times. In *STACS*, pages 380–391, 2011.
17. George B. Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, and Shmuel Zaks. Optimizing busy time on parallel machines. In *IPDPS*, pages 238–248, 2012.
18. Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, Fencol C.C. Yung, and Shmuel Zaks. Online optimization of busy time on parallel machines. *TAMC*, 7287:448–460, 2012.
19. Barbara B Simons and Manfred K Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM Journal on Computing*, 18(4):690–710, 1989.
20. Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *SODA*, pages 830 – 831, 2003.

7 Appendix: Busy Time Scheduling of Equal-length Jobs on a Bounded Number of Machines

Although it is impossible in the general case (see Lower Bounds, Section 6), in the case of $p_j = p$ we are able to compute a schedule which is $(1, O(1))$-approximate, i.e. with the optimal number of machines and $O(1)$ busy time vs. the busy time optimum. A lower bound shows that a $(O(1), 1)$-approximation is impossible to achieve, even in this scenario. In addition, we cannot get a result of the type $(1, O(1))$ for non-indentical job lengths (see Section 6).

Below we give an algorithm which computes the minimum number of machine and then generates a schedule. Of course, if we instead have a fixed machine budget then the algorithm can be used to either generate a feasible schedule or show it is infeasible. In this algorithm we let jobs be scheduled *cyclically* on machines: i.e. in the case $g = 1$, if we order jobs by increasing start time as jobs $J_0, J_1, J_2, \ldots$ then they would go onto machines $M_{0 \mod m}, M_{1 \mod m}, M_{2 \mod m}, \ldots$. For larger g, we would have the first g jobs on M_0, etc.

We now give an intuitive description of the algorithm. One hard problem when trying to generate a schedule satisfying the $(1, O(1))$ bound is getting any feasible schedule with the correct number of machines; recently, a nice $O(n^2)$ algorithm was developed by Ortiz and Quimper [16] but before that the best algorithm by Simons and Warmuth, based on the concept of "forbidden regions", was considerably more complicated [19]. However, all of the algorithms have the nice property that they generate *left-shifted* schedules (defined below), informally speaking starting jobs as early as possible. So instead of trying to solve the problem "from scratch" by a totally new method, we instead start with a left-shifted schedule and run a single-pass "pushing scanline" which, as it meets the initial start time of each job, begins pushing it along with the scanline until either a large number of jobs are clustered near the scanline (Step 3 (b)), or until a latest start time is met (Step 3(c)). We are then able to argue a crucial property (Prop 9) based on the intuition that if two jobs i and j both caused step 3 (c) to occur and job j starts after job i in the initial schedule, then their intervals must be disjoint; otherwise, the left-shifted initial schedule would only scheduled job j so much after the latest start time of job i if there were many jobs near job i keeping the machine busy, and in this case step 3 (b) would have been executed. (It turns out that $3p$ factor in step 3 (b) is crucial to the last part of this argument.)

We now make the intuition formal. Recall that w.l.o.g. we can assume that the release times and deadlines are integral.

Definition 1 *A schedule (s_j, m_j) using m machines is* left-shifted *when for every job j and time $r_j \leq t < s_j$, there exists a time $t' \in [t, t + p_j)$ such that $|\{k : [s_k, s_k + p_k) \ni t'\}| \geq mg$. Informally, this means that at every time t there are mg jobs blocking job j from starting.*

If a feasible schedule is not left-shifted, a single left-to-right pass over the jobs, swapping jobs to the earliest time t which violates the left-shifting property, will

give a feasible left-shifted schedule. However, the published feasibility algorithms generate schedules that minimize $\sum C_j$ (where C_j is the job completion time) [16], which is easily shown to imply the left-shifted property, and so running such a job-swapping pass is not actually necessary.

Algorithm Compact

1. Compute a feasible schedule S with the minimum number of machines by binary search ($0 \leq m_{opt} \leq n$), using a feasibility algorithm for the problem with mg identical machines, each with one processor. The best known algorithm is the $O(n^2)$ feasability algorithm of Lopez-Ortiz and Quimper [16].
2. Ensure that S is *left-shifted*. Let s_j^0 denote the start time of job j in S, and let $s_j := s_j^0$. Let $K := \emptyset$ and $P := \emptyset$.
3. For t from r_{min} to d_{max}: (*main loop*)
   (a) For every unscheduled job j, let $s_j := \max\{s_j^0, t\}$. Let U be the set of unscheduled jobs.
   (b) If $|\{j \in U : s_j \in [t, t+2p]\}| \geq mg$, run each job j in this set at time s_j. Let $K := K \cup \{[t, t+3p)\}$. We say these jobs were run in a *cluster*. Return to the main loop at $t := t + 2p$.
   (c) Otherwise if $t = u_j$ for some unscheduled job j, run each job in the set $\{j \in U : s_j \in [t, t+p]\}$ at its start time s_j. Return to the main loop at $t := t + p$. Let $P := P \cup \{j\}$.

In step 3 it is necessary to consider only $t \in \{u_j, s_j - 2p\}$, so we can run this step in linearithmic time. As a matter of terminology, we say we *executed* a step with an if-condition when the if-condition was true, i.e. we executed the body of the conditional.

We now begin the analysis. As in the previous section, it is trivial to verify that the resulting schedule contains all jobs, but now the feasibility becomes nontrivial since we do modify the start times of some of the jobs from the initial feasible schedule.

Proposition 8 *The generated schedule is feasible on $m = m_{opt}$ machines.*

Proof. We inductively verify that at every time t, after executing step 3 (a) there exists a feasible schedule starting jobs at time s_j. On the first iteration this is trivial, because after 3 (a) $s_j = s_j^0$. Observe that if in the previous iteration we executed either step 3(b) or step 3(c), then that iteration scheduled every job j with $s_j^0 < t$ so at time t no job's start time is modified. Thus from now on we may assume we did not execute either step 3(b) or 3(c) in the previous iteration.

Suppose for contradiction that at time t, after executing step 3 (a) there does not exist a feasible schedule with jobs starting at time s_j. Then by Proposition 1, we know at some time t' there are more than mg jobs j with $[s_j, s_j + p_j) \ni t'$. The only possible such t' is $t' = t + p - 1$, because the number of jobs containing every other (integer) point of time stayed constant. However, if this is true then there are more than mg jobs with start times in the window $[t-1, t+2p-1)$ and so in the previous iteration step 3(b) should have been executed. Contradiction.

Proposition 9 *The intervals $[r_j, d_j)$ for $j \in P$ are mutually disjoint. (It follows that $|P|p \leq OPT$.)*

Proof. By induction on $|P|$. Suppose we are at step 3(c) and job j is about to be added to the set P. Let job $i = \text{argmax}_{i \in P} d_i$, and suppose for contradiction that $r_j < d_i$. We know that $s_j^0 > d_i$, otherwise at step 3(c), time $t = s_i$, job j would have $s_j = \max\{s_j^0, s_i\} \leq d_i$ and thus would have been scheduled with job i. Because $r_i < d_i < s_j^0$, by the left-shifting property of the initial schedule we know that $|\{k : [s_k^0, s_k^0 + p) \ni d_i\}| \geq mg$. At time $t = s_i$ step 3(b) we would have had $s_k = \max(s_i, s_k^0) = s_k^0$ for each of the job k above, and thus there would have been more than mg jobs with start times in $[s_i, s_i + 2p_j)$ so step 3(b) would have been executed, running job i in a cluster. Then we would have job $i \notin P$, so by contradiction $r_j \geq d_i$. Since d_i is maximal, it follows that $[r_j, d_j)$ is disjoint from every $[r_p, d_p)$ for $p \in P$. The claim now follows by combining this with the inductive hypothesis.

Theorem 2. *Algorithm Compact is a 6-approximation for busy time, and generates a feasible schedule using m_{opt} machines.*

Proof. We only need to verify the busy time approximation ratio. Let T be the set of (m, t) where machine m is active at time t. Since every job is scheduled in either step 3 (b) or step 3 (c), we know that its execution time is either contained in one of the intervals in K, or in a window $[s_j, s_j + 2p)$ for $j \in P$, thus

$$\sum_m \mu(\{t : (m, t) \in T\}) \leq \sum_{I \in K} m\mu(I) + \sum_{j \in P} \eta_j \mu([s_j, s_j + 2p))$$

where η_j denotes the number of machines used in running the jobs in $[s_j, s_j + 2p)$. We bound the first sum on the rhs and part of the second sum by observe that cyclic scheduling gives a schedule every machine processes at least g jobs in an interval I (of length $3p$) for the first sum and for the second sum, all but at most 2 (the first and last machine used when scheduling jobs for this interval in step 3 (c)) of the η_j machines used process at least g jobs in each time interval $[s_j, s_j + 2p)$. Thus the load on those machines in those time intervals is at least $\min(1/3, 1/2) = 1/3$. To bound the remaining terms in the final sum, also by cyclic scheduling we know that for each $j \in P$ the latest job scheduled on the first machine used starts before every job scheduled on the last machine. Thus their busy windows overlap by at most p and their total busy time is at most $3p$. Therefore, combining this with the load bound we can bound the above by

$$3OPT + \sum_{j \in P} 3p \leq 6OPT$$

using $|P|p \leq OPT$.

We note that the above analysis gives a 5-approximation when $m_{opt} = 1$.

7.1 General Case, $g < \infty$

Combining with the bucketing algorithm given by Shalom et al [18] this gives a $O(\log \frac{p_{max}}{p_{min}})$-competitive online algorithm for busy time scheduling. More precisely, because the cost of their algorithm is bounded by 4 times the weight of the input jobs, and 1 times the $g = \infty$ lower bound, the approximation is $9 \log \frac{p_{max}}{p_{min}}$.

Running Algorithm Doubler offline and combining with the 3-approximation of Chang et al [5] gives a fast 7-approximation to the optimal busy time schedule. This is because the Greedy Tracking algorithm [5] takes as input a $g = \infty$ schedule using busytime T and outputs a schedule with cost at most $T + 2w(J)/g \leq T + 2OPT$ where $w(J)$ denotes the total processing time of all jobs. Since $T \leq 5OPT$ using our algorithm, the cost is bounded by $7OPT$.

If we are willing to grant the online algorithm a lookahead of $2p_{max}$ then we can get a constant factor online algorithm. We use our $g = \infty$ online algorithm to determine the time placement of jobs; this algorithm requires no lookahead so we now know the start time of jobs $2p_{max}$ ahead of the current time. We now run the offline machine-assignment algorithm in windows of the form $[kp_{max}, (k+2)p_{max})$ for $k \in \mathbb{N}$. We can bound the cost of even k by $5OPT + 2w(J_0)/g$ where $w(J_0)$ is the total processing time of jobs run in windows with even k; adding the matching term for odd k shows that this gives a fast $2 * 5 + 2 = 12$ approximation.

8 Appendix: Offline Algorithm for Bounded Number of Machines (with proofs)

In this section we will use the fact that scheduling jobs on a minimum number of machines with integer release times and deadlines and with $p = p_j = 1$ is trivial offline. For a fixed m, it is well-known, by a swapping argument that an EDF (earliest-deadline first) schedule, i.e. one given by running at each time up to m of the jobs with earliest deadlines, gives a feasible schedule iff the input instance is feasible. Computing the minimum m can be done by binary search in $\log n$ steps.

We would like to describe some intuition before launching into the formal analysis. As before, we use something like a "pushing scanline" approach, starting from a left-shifted schedule and starting a group of jobs whenever a large number have been pushed together. To make this approach have bounded busy time usage, we need to bucket jobs by similar lengths and use the approach on a per-bucket basis, but this alone cannot attain our desired performance ratio, because we may need, for busy time purposes, to group some long jobs with some short jobs. (If we are never willing to group together jobs of different lengths then it is clear we cannot get better than a $O(g)$ busy time on difficult input instances.) Therefore, in each bucket, when a job does not nicely group together with other jobs of the same length, we temporarily drop it. A second "clean-up pass" (step 3 below) runs the remaining jobs using an algorithm which has good busy-time performance but a priori unbounded machine usage. By arguing that we drop few jobs with overlapping availability times from each bucket, it is then

possible to bound the machine usage (see proof of Theorem 4); this is not dissimilar in spirit to the disjointness argument from the previous section, but is complicated by all of the rounding that occurs in the bucketing process.

We now present a $(O(\log p_{max}/p_{min}), O(1))$-approximation algorithm for the general problem. Fix a constant[3] $\alpha > 1$.

1. Bucket jobs by processing time increasing exponentially by α, so the buckets contain jobs of processing time in the intervals $[p_{min}, \alpha p_{min}), [\alpha p_{min}, \alpha^2 p_{min}), \ldots, [\alpha^{q-2} p_{min}, \alpha^{q-1} p_{min}), [\alpha^{q-1} p_{min}, \alpha^q p_{min}]$ where $q = \left\lceil \log_\alpha \frac{p_{max}}{p_{min}} \right\rceil$.
2. For each bucket B_i
   (a) Let p be the supremum of the processing times of jobs in this bucket. We round job availability constraints down to multiples of p, so $r'_j = p\lfloor r_j/p \rfloor, u'_j = p\lfloor u_j/p \rfloor$, and $p'_j = p$. This is a unit job scheduling problem after we rescale by a factor of p.
   (b) We generate a left-shifted feasible schedule (referred to as the *initial schedule*) for the rounded (r'_j, d'_j, p'_j) instance using the minimum number of machines m. Let s^0_j be the start time of job j in this schedule.
   (c) Execute Algorithm RunHeavy.
   (d) Let U'_i denote the set of jobs unscheduled in B_i after running Algorithm RunHeavy.
3. Now let U'' be the union of the U'_i for all buckets, and schedule the jobs in U'' by the 3-approximation of Chang et al [5] upon a new set of machines (we will bound the number needed).

Algorithm RunHeavy

1. Let U initially be the set of all jobs in the bucket. Split machines into groups M_1 and M_0; we will require at most m machines in each group (see Proposition 14 below).
2. For[4] $t = kp$ from r'_{min} to u'_{max}:
   (a) Let $J_t = \{j \in U : s^0_j = t\}$. Let $k_1 = \lfloor |J_t|/g \rfloor$ and run k_1 groups of g jobs from this set on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s^0_j, r_j)$. Remove these jobs from U.
   (b) Let $J'_t = \{j \in U : s^0_j \leq t \leq u'_j\}$. Let $k_2 = \lfloor |J'_t|/g \rfloor$ jobs, and run k_2 groups of g jobs from this set on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s^0_j, r_j)$. Remove these jobs from U.

For space reasons the proofs have been moved to the Appendix. We note that although the time complexity of our algorithms depends on $\log \frac{p_{max}}{p_{min}}$, since we assume inputs are integral $\log_2 p_{max}$ is bounded by the number of bits in the

[3] The choice of α will involve a trade-off between the constants in the busy time and machine usage ratio.

[4] For ease of presentation, we presented this as a for-loop over time; however, to ensure that it is polynomial time, observe that we only actually need to run iterations when $t = s^0_j$ for some job so the actually loop is linearithmic.

processing time of some job, hence our algorithm is polynomial-time in the size of the input. We also note that the algorithm picks starting times respecting $[r_j, d_j]$ constraints, from the following two (trivial) propositions:

Proposition 10 *Fix a bucket. Suppose $r'_j \le s'_j \le u'_j$. Then if we let $s_j = \max(s'_j, r_j)$, we have $r_j \le s_j \le u_j$.*

Proof. That $r_j \le s_j$ is immediate from its definition. We must have one of $s_j = s'_j$ or $s_j = r_j$. If $s_j = s'_j$, then $s'_j \le u'_j = p\lfloor u_j/p \rfloor \le u_j$. If $s_j = r_j$, then by assumption (see the introduction) $s_j = r_j \le u_j$.

Proposition 11 *Start times s_j selected by this algorithm respect release time and deadline constraints, i.e. $r_j \le s_j \le u_j$.*

Proof. This is already verified for start times chosen by Doubler. This just needs to be verified for steps 2(a) and 2(b) of RunHeavy, at which point for every job j being run, we trivially have $r'_j \le t \le u'_j$, using the fact $r'_j \le s^0_j \le u'_j$ since s^0_j comes from a feasible schedule. Proposition 10 then implies the result.

Proposition 12 *Fix a bucket. Let m be as above, and let m_{opt} (for this lemma only) denote the minimum number of machines required to process the jobs in this bucket with their true (r_j, d_j, p_j) values. Then $m \le \lceil \alpha \rceil m_{opt}$.*

Proof. Let s_j denote the start times of jobs in the schedule of the (r_j, d_j, p_j) input on m_{opt} machines, and let $s'_j = p\lfloor s_j/p \rfloor$, so $s'_j \in [r'_j, u'_j]$. We claim that the (s'_j) determine a feasible schedule for the (r'_j, d'_j, p'_j) using at most $\lceil \alpha \rceil m_{opt}$ machines; this means (e.g. by Proposition 1) that we have to verify that

$$|\{i : s_i \in [t, t+p)\}| = |\{i : s'_i = t\}| \le \lceil \alpha \rceil m_{opt}.$$

for all $t = kp$. Fix t. Using Proposition 1 and $p/\alpha \le p_j$, we know that in the initial schedule for any t'

$$|\{j : [s_j, s_j + p/\alpha) \ni t'\}| \le |\{j : [s_j, s_j + p_j) \ni t'\}| \le m_{opt}.$$

Now if we consider points t' of the form $t + \ell p/\alpha - \epsilon$ for $1 \le \ell \le \lceil \alpha \rceil$ and ϵ sufficiently small (more precisely $\epsilon < \inf_{s_j \in [t, t+p)} t + p - s_j$), the time interval of every job with $s_i \in [t, t+p)$ will contain one of these points, so there can be at most $\lceil \alpha \rceil m_{opt}$ such jobs total.

Proposition 13 *Fix a bucket. At every time t, we have $k_2 \le 1$, $k_1 + k_2 \le m$ and after executing step 2 (b) we have $|J'_t \cap U| < g$.*

Proof. By induction on t. Denote by U_1 the set U after executing step 2 (a). Observe that $J'_t \subset (J'_{t-p} \cap U_1) \cup \{j \in U_1 : s^0_j = t\}$, and the two sets unioned on the rhs are both of cardinality less than g, so $|J'_t| < 2g$ and $k_2 \le 1$. If $k_1 = m$, so the maximum mg jobs were run at time t in the initial schedule, then $\{j \in U_1 : s^0_j = t\} = \emptyset$ so $|J'_t| < g$ and $k_2 = 0$. Henceforth we must only consider the case when $k_1 < m$. If $|J'_t| < g$ then we are done. Otherwise, $|J'_t| \ge g$, then at step 2 (b) g of these jobs will be scheduled and removed from U, so $|J'_t \cap U| < 2g - g = g$ and $k_2 = 1$, hence $k_1 + k_2 \le m$.

Proposition 14 *Fix a bucket. Machine groups M_1 and M_0 each require at most m machines.*

Proof. W.l.o.g. we prove this only for M_0. Observe that at time $t = 2k'p$ for some $k' \in \mathbb{Z}$, the start times of the jobs we run at step (2) of algorithm RunHeavy lie within $[2k'p, 2k'p + p)$, hence the jobs execution windows $[s_j, s_j + p_j)$ are contained in $[2k'p, 2k'p + 2p)$. Thus when running each iteration of the loop at step (2) of RunHeavy with $k = 2k'$, every machine in M_0 has completed all of its previously scheduled jobs and is now idle. At this point, to schedule the jobs run at steps 2(a) and 2(b) we will need at most $k_1 + k_2 \leq m$ idle machines, by Proposition 13; hence we need no more than m machines in M_0.

Proposition 15 *Fix a bucket. If $i \in U'$, then $|\{j \in U' : [r'_j, u'_j] \ni s^0_i\}| < g$.*

Proof. Because job i was not run in step 2 (a) at time s^0_i, we in particular know that fewer than mg jobs were scheduled in the initial schedule at time s^0_i. Then by the left-shifting property, there exists no job j such that $r'_j \leq s^0_i$ and $s^0_j > s^0_i$. By Proposition 13, there exist fewer than g jobs in U' such that $s^0_j \leq s^0_i \leq u'_j$.

Proposition 16 *Fix a bucket. For any t, $|\{j \in U' : [r'_j, u'_j] \ni t\}| < 2g$.*

Proof. Let s^0_α be maximal such that $s^0_\alpha \leq t$, and let s^0_β be minimal such that $s^0_\beta > t$. For any j such that $[r'_j, u'_j] \ni t$, either $s^0_j \leq s^0_\alpha$ or $s^0_j \geq s^0_\beta$, so either $s^0_\alpha \in [s^0_j, t] \subset [r'_j, u'_j]$ or $s^0_\beta \in [t, s^0_j] \subset [r'_j, u'_j]$. By Proposition 15, fewer than g jobs can contain each of s^0_α and s^0_β.

Proposition 17 *Fix a bucket. For any t, $|\{j \in U' : [r_j, d_j) \ni t\}| < 4g$.*

Proof. Let j be such a job and let $t' = p\lfloor t/p \rfloor$. By monotonicity of floor, $t' \in [r'_j, p\lfloor d_j/p \rfloor] \subset [r'_j, u'_j + p]$. Therefore $[r'_j, u'_j]$ contains at least one of $\{t', t' - p\}$, so the result follows by Proposition 16.

Theorem 3. *Given an input of fixed interval jobs which can be feasibly scheduled on γ processors, the algorithm of Chang et al uses at most 2γ processors [5].*

Proof. We briefly recall the algorithm: define a *track* to be a disjoint union of job intervals; then the algorithm repeatedly picks a track of maximal measure (length) from the set of unscheduled jobs and schedules the jobs in this track onto the next processor.

Suppose that this algorithm uses β processors. Let j be an arbitrary job in the final track occupying the time interval $[s_j, s_j + p_j)$. Let T be a track assigned to any other processor. We claim that $T \ni s_j$ or $T \ni s_j + p_j$. Suppose otherwise; then $\mu(T \cap [s_j, s_j + p_j)) < p_j$, so the track $(T \setminus [s_j, s_j + p_j)) \cup [s_j, s_j + p_j)$ is a valid track of greater measure, violating maximality of T.

Finally, by the pigeon hole principle either $|\{T \ni s_j\}| \geq \beta/2$ or $|\{T \ni (s_j + p_j)\}| \geq \beta/2$, so the result follows from Proposition 1.

Theorem 4. *The resulting scheduling requires at most* $q(2\lceil\alpha\rceil m_{opt} + 8)$ *machines.*

Proof. It follows from Proposition 17 and Proposition 1 that scheduling the jobs in U'' requires at most $4qg$ processors, i.e. $4q$ machines, so by Theorem 3 the schedule actually generated for the jobs in U'' will require at most $8q$ machines. Thus combining with Proposition 12 and Proposition 14 which tell us that RunHeavy uses at most $2\lceil\alpha\rceil m_{opt}$ machines for each bucket, in total the schedule requires $q(2\lceil\alpha\rceil m_{opt} + 8)$ machines.

Theorem 5. *The resulting schedule has busy time cost at most* $(2\alpha + 1)OPT$.

Proof. Similar to the previous section, observe that the load on a machine used by RunHeavy, when active, is at least $1/2\alpha$: when RunHeavy at time t runs a set of g jobs on a machine at time, we know by our rounding mechanism that for a job j, $r_j \leq s_j + p$, so every job fits in the window $[t, t + 2p)$ and is of length at least p/α. So by the load bound, the cost of the jobs scheduled by the calls of RunHeavy is upper bounded by $(2\alpha)w(J_c)/g$ where J_c denotes the set of jobs scheduled by RunHeavy. The algorithm of [5] has cost bounded by $2w(U)/g + OPT$, so if J is the set of all jobs the cost of our schedule is bounded by $\max(2\alpha, 2)w(J)/g + OPT \leq (2\alpha + 1)OPT$.

Picking a particular α, for example $\alpha = 2$, gives the claimed result in the introduction.

9 Appendix: Online Algorithm

We now describe the algorithm in full detail. Because our algorithm is online, we must be careful to precisely describe the flow of information, so as to ensure that we compute the right amount of lookahead for the total algorithm. However, a simple *compositional principle* makes this feasible: suppose x_t represents the input to the online algorithm at step t, and $X_t = (x_1, \ldots, x_t)$. Let f represent an online algorithm which takes input from x with lookahead ℓ and produces output f_t at time t; then mathematically, f_t is a function of $X_{t+\ell}$, which we denote as $f_t = f_t(X_{t+\ell})$. Let g represent an online algorithm which needs both the original input and the value of f_t with lookahead ℓ'; mathematically, g_t is a function of $f_{t+\ell'}$ and $X_{t+\ell'}$, so $g_t = g_t(f_{t+\ell'}, X_{t+\ell'}) = g_t(f_{t+\ell'}(X_{t+\ell+\ell'}), X_{t+\ell'})$; since $X_{t+\ell'}$ is contained in $X_{t+\ell+\ell'}$, we have verified that g_t needs lookahead $\ell + \ell'$ to process the original input X. Thus when we compose online algorithms in this fashion, the total lookahead required by the algorithm adds.

Now we describe the algorithm as a *composition* of phases, each with their own lookaheads, and the lookahead of the algorithm as a whole is the sum of these individual lookaheads. The list below consists of phases, which are in turn composed of normal sequential steps.

1. (Lookahead 0) As before we can bucket jobs; since we assumed that our inputs were integral, we could bucket the processing times as $[1, \alpha), [\alpha, \alpha^2), \ldots$

and so on. For convenience, since we assume all input is integral, we actually bucket as $[1, \lfloor\alpha\rfloor), [\lceil\alpha\rceil, \lfloor\alpha^2\rfloor), \ldots$.

2. (Lookahead 0) Fix a total ordering $<_\omega$ of jobs, such that if job i, j have $d'_i \leq d'_j$, then $i \leq_\omega j$. Whenever we refer to selecting jobs EDF, it means select the first jobs according to this ordering. (So we have broken ties arbitrarily, but in a consistent fashion throughout the algorithm.)
3. (Lookahead p) For each bucket, in parallel:
   (a) (Lookahead p) Let p be the maximum possible job size in the bucket. We round job availability constraints down to multiples of p, so $r'_j = p\lfloor r_j/p\rfloor, u'_j = p\lfloor u_j/p\rfloor$, and $p'_j = p$. Note that it requires p lookahead to round the input instance into the unit job instance, since r'_j may be as small as $r_j - p - \epsilon$ for any ϵ.
   (b) (Lookahead 0) This is a unit job scheduling problem after we rescale by a factor of p. We use, with slight modification, the algorithm of [7] (treating processors as machines) to compute online, with no lookahead, a schedule S of the rounded jobs which uses at most $\lceil em\rceil$ processors, where m is the minimum number of machines necessary to schedule the input $\{(r'_j, u'_j, p'_j)\}$. The slight modification is that whenever the original algorithm would have opened η processors at time t, the modified algorithm opens $\lceil\eta/p\rceil$ machines, giving $p\lceil\eta/p\rceil$ processors; at every point in time this opens more processors than the standard algorithm, so the generated schedule is still feasible. Let s^0_j be the start time of job j in this generated schedule, and let $m_t(B_i)$ be the number of machines activated. The start times are guaranteed to be at times of the form kp for $k \in \mathbb{Z}$.
4. (Lookahead 0) Run FixStartTimes, using the outputs of the previous phase. (This algorithm plays the role of *both* RunHeavy and Doubler in previous sections.) This algorithm "marks for execution" a set of jobs E, which are informally the jobs Doubler has chosen to execute; their start times are already determined, but these are passed to the next phase to choose a machine for them to run on.
5. (Lookahead $2p$) To determine machine assignments for the jobs in E, we use the use the technique from subsection 7.1. To recall, we assign every job in E (now with fixed start time) to some window containing it of the form $[k, k + 2p_{max})$, and then use the machine-selection algorithm of Chang et al [5] to do machine selection in each window; we share machines between even k and between odd k.

Algorithm FixStartTimes

1. Arbitrarily label the buckets as $B_1, B_2, B_3, \ldots$. Let $U(B_i) = \emptyset$. Let $P = \emptyset$. As in RunHeavy (step 1), create (initially empty) sets of machines $M_1(B_i)$, $M_0(B_i)$, $M'_1(B_i), M'_0(B_i)$ for every bucket. As in Doubler, let $T = \{\}$. We also initialize two sets of jobs, $D = \{\}$ and $E = \{\}$.
2. For[5] t from r'_{min} to u'_{max}:

[5] As before, the set of t which we actually need to consider is much smaller, polynomial size, and we write the code this way only for ease of presentation; it is especially natural to write an online algorithm this way.

(a) For each bucket B_i:
   i. Add to $U(B_i)$ all jobs with $s_j^0 = t$ in bucket i.
   ii. Remove from $U(B_i)$ any jobs in $D(B_i) \cup E$.
   iii. If $t = kp$ where $p = p(B_i)$ is the max possible processing time in bucket B_i, execute the following steps[6]:
      A. Let $J_t = \{j \in U(B_i) : s_j^0 = t\}$. Let $k_1 = \lfloor |J_t|/g \rfloor$ and run k_1 groups of g jobs, selected EDF from this set on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s_j^0, r_j)$. Remove these jobs from $U(B_i)$ and mark them as *scheduled*.
      B. Let $J'_t = \{j \in U(B_i) : s_j^0 \leq t \leq u'_j\}$. Let $k_2 = \lfloor |J'_t|/g \rfloor$ jobs, and run k_2 groups of g jobs from this set selected EDF on the group of machines $M_{k \mod 2}$ with start times $s_j = \max(s_j^0, r_j)$. Remove these jobs from $U(B_i)$ and mark them as *scheduled*.

(b) Let U_t be the set of unscheduled[7] jobs j which have $r'_j \leq t$ and are not in $\bigcup_i D(B_i) \cup E$. Let $U_t(B_i)$ be those jobs in U_t from bucket B_i.

(c) If $t = u'_k$ for some $k \in U_t$, then pick such a k with p'_k maximal. If furthermore, $[t, t + 2p'_k) \not\subset T$, then set $T := T \cup [t, t + 2p_k)$ and $P := P \cup \{k\}$.

(d) For each bucket B_i:
   i. If $[t, t + p(B_i)) \subset T$:
      A. Let $J''_k(B_i)$ be $J'_t \cap U(B_i)$ extended with up to g jobs from $U_t(B_i)$, such that $|U_t(B_i) \setminus J''_k(B_i)| \equiv 0 \mod g$. Let $J'_k(B_i) = U_t(B_i) \setminus J''_k(B_i)$.
      B. Set $D(B_i) := D(B_i) \cup J'_k(B_i)$. Set $E := E \cup J''_k(B_i)$, and for every job $j \in E$ set $s_j := \max(t, r_j)$.
   ii. If $t = kp(B_i)$ for some $k \in \mathbb{Z}$: take[8] up to $m_t(B_i)g$ jobs from $D(B_i)$, choosing earliest-deadline first; for each job j in this set, let its start time be $s_j = \max(t, r_j)$ and assign it to machine group $M'_{k \mod 2}$. Remove these jobs from $D(B_i)$.

In total the algorithm requires $3p_{max}$ lookahead. The same argument as for Doubler shows that it schedules all the jobs:

Proposition 18 *At no point in time t' does U_t contain a job j with $u'_j < t'$.*

Proof. By contradiction; suppose there exists such a job j. Then at time $t = u'_j$, step 2.c we are ensured that either a job k was chosen with $p'_k \geq p'_j$ and $k \in P$ and that after this step $[t, t+2p'_j) \subset [t, t+2p'_k) \subset T$, or if no such job was chosen that $[t, t + 2p'_j) \subset T$. Either way, we are ensured that at step 2.d.i.B, this job must have been added to E, because since $t = u'_j$ and $s_j^0 \leq u'_j$ by feasibility, job j necessarily lies in J'_t, and by assumption it lies in $U_t(B_i)$, so it lies in $J'_t \cap U(B_i)$, hence $J''_k(B_i)$.

[6] These are steps 2(a) and 2(b) of RunHeavy with $U(B_i)$ replacing U
[7] This means just that they have not been marked as scheduled in step 2.a
[8] i.e. take $\min(m_t(B_i)g, |D(B_i)|)$ jobs

Proposition 19 *When a job j is added to $D(B_i)$ in step 2.d.i.B at time t, we have $s_j^0 > t$.*

Proof. By contradiction; if $s_j^0 \leq t$ then , since by the previous proposition $t \leq u_j$, we have that $j \in J'_t$ hence it must be that $j \in J''_k(B_i)$ instead of $J'_k(B_i)$, hence j is not in $D(B_i)$.

Proposition 20 *Fix a bucket B_i. After each execution of step 2.d.ii, at time t, the set of jobs* assigned to *machine groups M'_0 and M'_1 is a superset of the set $S_t = \{j \in D(B_i) : s_j^0 \leq t\}$.*

Proof. By induction on time. Supposing the IH, we need verify this only for new elements of S_t (formally, elements of $S_t \setminus S_{t-1}$). By the previous proposition, these are jobs which were already in $D(B_i)$ before time t, hence their being added to S_t means that $s_j^0 = t$. There can be at most $m_t(B_i)$ such jobs; by virtue of having been chosen EDF from a larger set of jobs, the subset of these jobs which have not already been scheduled on M'_0 or M'_1 are ensured to be subset of the up to $m_t(B_i)$ jobs chosen EDF by step 2.d.ii.

Proposition 21 *Fix a bucket B_i. At the end of the loop iteration for time t, the union W_t of the following sets of jobs:*

- *The set of jobs executed on M_0 or M_1*
- *The set of jobs assigned to M'_0 orM'_1*
- *Jobs in E*
- *Jobs in $U_t(B_i) \setminus (E \cup D(B_i))$.*

are a superset of the set of jobs started by this point in the online unit job schedule, i.e. the set $K_t = \{j \in B_i : s_j^0 \leq t\}$.

Proof. First observe, by the previous Proposition, that every job in $K_t \cap D(B_i)$ is assigned to one of M'_0 or M'_1. Thus it remains to show that $W_t \setminus D(B_i)$ contains the jobs in $K_t \setminus D(B_i)$. By induction on t. Consider a job $j \in K_t \setminus D(B_i)$. If $s_j^0 < t$, then by the IH this job was in $W_{t-1} \setminus D(B_i)$ (for a possibly smaller set $D(B_i)$ at time $t-1$). Observe that only way job j is moved from W_{t-1}, i.e. lies in $W_t \setminus W_{t-1}$ is if the job is moved into $D(B_i)$ at step 2.d.i.B, but since $j \notin D(B_i)$ this is impossible. It remains to consider when $s_j^0 = t$. Once again, observe that for any such job j which does not lie in $D(B_i)$, it is either: 1. already contained in E or 2. added to $U_t(B_i)$, and possibly moved into set E at step 2.d.i.B. Either way, we find that $j \in W_t \setminus D(B_i)$.

Proposition 22 *At every time t', every job $j \in E$ has $r'_j \leq t'$, and the same holds for every $j \in D(B_i)$ for all i.*

Proof. The arguments are the same, so we present it for E. Every job in E at time t' was added at an earlier time t at step 2.d.ii.B. At this step, we add to E the elements of $J''_k(B_i) \subset U_t$, and every job $j \in U_t$ has $r'_j \leq t$ ($\leq t'$), by definition of U_t (step 2.b).

Proposition 23 *The online algorithm schedules all jobs, generating a feasible schedule.*

Proof. By Proposition 10, verifying our job's start times obey release time and deadline constraints, when they are of the form $s_j = \max(r_j, s'_j)$ for some s'_j, is reduced to verifying that start time s'_j obeys the rounded r'_j, u'_j constraints. For start times chosen in step 2.a.iii, this argument is unchanged from Proposition 11. For start times of jobs in E chosen in step 2.d.ii.B, we must verify that $r'_j \leq t \leq u'_j$. That $r'_j \leq t$ is Proposition 22 above. To show $t \leq u'_j$, suppose for contradiction that $t > u'_j$, then at time $u'_j < t$, step 2.d.iii this job would already have been added to either $D(B_i)$ or E, hence it cannot be in U_t; contradiction.

Finally we consider start times of jobs selected in step 2.d.iv. Once again by Proposition 22 we have $r'_j \leq t$ for all $j \in D(B_i)$ scheduled at this step, with the loop at time t. To show $t \leq u'_j$, observe that by Proposition 20 that every job in $D(B_i)$ is scheduled at step 2.d.ii at a time $t \leq s^0_j \leq u'_j$.

(The following proposition is not needed, but is an easier case of the one which follows.)

Proposition 24 *Suppose an input instance with $g = \infty$ can be scheduled with busy time $busy_{opt}$ with start times s_j, then the rounded input instance with $s'_j = p_{max}\left\lfloor \frac{s_j}{p_{max}} \right\rfloor$, r'_j and u'_j down to a multiple of p_{max} by the same formula, and $p'_j = p_{max}$ can be scheduled with busy time at most $(1 + 2\frac{p_{max}}{p_{min}})busy_{opt}$.*

Proof. Henceforth $p = p_{max}$. Let $T = \bigcup_j [s_j, s_j + p_j)$ so $\mu(T) = busy_{opt}$. Split T into connected components, so $T = [\alpha_1, \beta_1) \cup \cdots [\alpha_r, \beta_r)$ and every interval $[s_j, s_j + p_j)$ is contained in one of these components. After rounding, the jobs occupying time interval $[\alpha_i, \beta_i)$ are now contained in the interval $[p\left\lfloor \frac{\alpha_i}{p} \right\rfloor, p + p\left\lfloor \frac{\beta_i}{p} \right\rfloor)$, which has length at most $\beta_i - \alpha_i + 2p$. Since $\beta_i - \alpha_i \geq p_{min}$, the ratio of lengths is at most

$$\frac{\beta_i - \alpha_i + 2p}{\beta_i - \alpha_i} \leq 1 + 2\frac{p_{max}}{p_{min}}.$$

Proposition 25 *Suppose we have an input instance with $g = \infty$ like before with optimal busy time $busy_{opt}$. After performing bucketed rounding (step 3.a), the optimal busy time for the new input instance has busy time at most $(1 + 2\alpha)busy_{opt}$.*

Proof. Once again, pick an busytime optimal schedule with start times s_j and let $T = \bigcup_j [s_j, s_j + p_j)$, and perform bucketed rounding so $s'_j = p\left\lfloor \frac{s_j}{p} \right\rfloor$ where p is the length of the largest job in the bucket. Decompose T into connected components $[\alpha_1, \beta_1) \cup \cdots \cup [\alpha_r, \beta_r)$. For an interval $[\alpha_i, \beta_i)$, the B_i be the bucket containing the job with largest processing time p in this interval. Let p' be the length of the longest job in this bucket. After rounding, the jobs formerly contained in this interval must now be contained within the interval $[\alpha_i - p', \beta_i - p')$ and

$$\frac{\beta_i - \alpha_i + 2p'}{\beta_i - \alpha_i} \leq 1 + 2\frac{p'}{p} \leq 1 + 2\alpha.$$

Theorem 6. *The online algorithm requires* $q(16+4\lceil e\lceil\alpha\rceil m_{opt}\rceil)$ *machines. Here* $q = \lceil \log_\alpha p_{max}\rceil$*, which is the maximum number of buckets used.*

Proof. First we bound machine usage for jobs in E. Fix a bucket B_i. At each time t in step 2.d.i.B, we run at most $2g$ jobs in E from this bucket, and this only occurs at times t which are either a multiple of the form kp ($p = p(B_i)$, $k \in \mathbb{Z}$) , or at which a new job has been added to P, and no job was added to P since the last multiple kp(with processing time at least $p(B_i)/2$); otherwise, because jobs are released at multiples of $p(B_i)$, there are no new jobs in $U_t(B_i)$ to run. Thus, with rounded start times, at most $2(2g) = 4g$ jobs can overlap at the same time, and this can only be achieved in the overlap created when a job is added to P at a time which is not a multiple of kp. Thus, after unrounding start times (i.e. letting $s_j = \max(s'_j, r_j)$, which can increase s_j by at most p where p is the maximum length of a job) and unrounding processing times, at most $8g$ jobs can overlap at any point in time, so by Proposition 1 at most 8 machines are needed in the machine-optimal schedule to run these jobs. Summing over the buckets gives a bound of $8q$ for the machine-optimal schedule, and finally by applying Theorem 3 twice, once for scheduling in step (4) with even k and once for odd k, the schedule actually generated for the jobs in M will require at most $8q + 8q = 16q$ machines.

Next we bound machine usage for step (2b) of FixStartTimes, the component of the algorithm informally corresponding to RunHeavy. First, consider first a particular bucket. Taking into account that our initial schedule (given by the s^0_j) in each bucket uses at most $\lceil em\rceil$ machines, where m is the minimum number of machines to schedule the rounded input (r'_j, u'_j, p), we observe that we can apply Proposition 12 and (the argument of[9]) Proposition 14 to show that the schedule uses in total $2q\lceil e\lceil\alpha\rceil m_{opt}\rceil$ machines.

Finally, bounding machine usage for jobs run in step (2.d.ii), i.e. those from $D(B_i)$, is similar: once again by Proposition 12, the schedule with rounded start times takes at most $q\lceil e\lceil\alpha\rceil m_{opt}\rceil$ machines in each of M'_0, M'_1, and unrounding does not affect this (this argument has been made before, but explicitly: because we alternate machine groups and unrounding moves start times by only at most p, jobs scheduled on different iterations of the loop can never overlap.) This gives a bound of $2q\lceil e\lceil\alpha\rceil m_{opt}\rceil$ for this part of the algorithm. Then in total the algorithm uses $q(16 + 4\lceil e\lceil\alpha\rceil m_{opt}\rceil)$ machines.

Theorem 7. *The online algorithm requires at most* $(20+42\alpha)busy_{opt}$ *busy time.*

Proof. To bound the busy time for step 2.a.iii of FixStartTimes, i.e. the RunHeavy component of the algorithm, is just a load-bound argument exactly the same as found in the proof of Theorem 5; the cost is bounded by $2\alpha w(J_c)/g$ where J_c is the set of jobs scheduled in step (2b) (i.e. those not in M). To bound the busy time for step 2.d.ii is similar, once we note that this step, like 2.a.iii, always schedules a multiple of g jobs. This is trivially verified by induction: in

[9] Technically we must change the names of some variables in Proposition 14 and the proposition leading up to it, but nothing more complicated is required.

step 2.d.i.B the set of jobs added to $D(B_i)$ always has size a multiple of g (see step 2.d.i.A), so the number of jobs run, $\min(m_t(B_i)g, |D(B_i)|)$, is a multiple of g hence the size of $D(B_i)$ after removing these jobs is still a multiple of g. Thus we can apply the same load-bound argument as for RunHeavy to find that the cost is bounded by $2\alpha w(J_d)/g$ where J_c is the set of jobs run in step 2.d.ii.

To bound the cost of jobs in E, informally the Doubler component of the algorithm, we use a simple simulation argument: the rounded start times selected in step 2.i.B (i.e. the rounded start time of a job is t for the loop iteration it was started in) are the same as those that would be selected by algorithm Doubler if run on the jobs in E *with the rounded constraints* (r'_j, u'_j, p'_j). To verify this, we first need to verify that any job k picked in step 2.c to add to P is later added to E; however, because $t = u'_k$ we know that $s^0_k \leq t \leq u'_k$ so $t \in J'_t \cap U(B_i)$ where B_i is the bucket containing job k, it is ensured that at step 2.d.i.B at time t this job is in $J''_k(B_i)$, hence added to E. We next observe that if we remove step 2.b from Doubler, and add to step (c) a condition that we only modify T and P if $[u_j, u_j + p_j) \not\subset T$, then algorithm Doubler behaves identically: in the modified algorithm, either step (c) is a no-op on this iteration of the loop, or if it isn't a no-op (so $[u_j, u_j + p_j) \not\subset T$, then because $[u_j, u_j + p_j) \not\subset T$, this job j would still have been chosen by step 2(c) of the original algorithm, and those jobs that would have been scheduled in step 2(b) are still scheduled in step 2(d) at the same start times. Now we can directly observe that steps 2 (c) and 2 (d) of this modified Doubler act identically (by choosing start times and modifying T and P) when fed input set E, as steps 2.c and 2.i.B do when running our algorithm.

Thus, combining the guarantee of Theorem 1 with the rounding-cost bound from Proposition 25, we get a bound of $5(2\alpha+1)busy_{opt}$ on the busytime of the *rounded schedule* of jobs in E running on a processor with $g = \infty$, where each job starts at the time it was added to E and takes up p'_j time. If we denote these rounded start times as s'_j, and let $T' = \bigcup_{j \in E}[s'_j, s'_j + p'_j)$, then $\mu(T')$ is the busytime we just bounded. Decompose T' into connected components, so $T' = [\alpha_1, \beta_1) \cup \cdots \cup [\alpha_r, \beta_r)$. Since we let $s_j = \max(s'_j, r_j)$ in step 2.d.B (recall $s'_j = t$ here), after unrounding the jobs in interval $[\alpha_i, \beta_i)$ now are contained within an interval $[\alpha_i, \beta_i + p)$ where p is the maximum of the p'_j for jobs in this interval. The same type of ratio bound as in Prop 25 observes that

$$\frac{\beta_i + p - \alpha_i}{\beta_i - \alpha_i} \leq 2,$$

so the measure of the $g = \infty$ cost of running the jobs (henceforth denoted as $\mu(T)$, where T is the times the machine is on) at their unrounded start times is at most $10(2\alpha + 1)busy_{opt}$. Then just as in subsection 7.1, when we apply the algorithm of Chang et al [5] for even k and for odd k in step 4, we get that the cost is bounded by $2\mu(T)+2w(E)/g$. Finally, we see the total busy time is upper bounded by

$$\begin{aligned} &2\mu(T) + 2w(E)/g + 2\alpha w(J_d)/g + 2\alpha w(J_c)/g \\ &\leq [20(2\alpha + 1) + 2\alpha]busy_{opt} = [42\alpha + 20]busy_{opt} \end{aligned}$$