Busy Time Scheduling on a Bounded Number of Machines (Extended Abstract)*

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* [4,12,8,11,15,5]. 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. [11,5,4] . 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

* Full version: http://math.mit.edu/~fkoehler/busytime.pdf. Research supported by CCF 1217890 and CNS 1262805.

(OADMs) [8], and the application of busy time models to optical network design has been extensively outlined in the literature [8,9,18,1].

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 [18] 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 [13], 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 [8] 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 given by Khandekar et al [11]. 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 [8]. 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 [17]. 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 [7,10] claims a 2-competitive online algorithm when $g = \infty$, but it is incorrect; see Fig. 1. *Independently and simultaneously to us*, Ren and Tang [16] recently studied the online problem when $g = \infty$ as well (see next section). They proved the same lower bound on the competitive ratio of this problem as we do and gave a slightly worse upper bound, $4 + 2\sqrt{2}$. They also analyzed a version of the problem where job lengths are unknown to the scheduler and proved a strong lower bound in this setting.

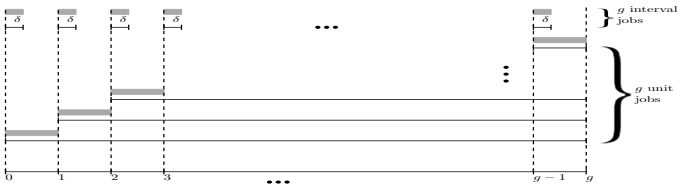


Figure 1: Counter-example to online algorithm of [7]. The optimal solution delays all the flexible unit jobs to the end and gets a busy time cost of $1 + g\delta$ rather than g. Setting $\delta = \frac{1}{g}$ gives the gap. The figure shows the schedule produced by the online algorithm with a cost of g.

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 $\varphi = (1 + \sqrt{5})/2$.
- 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 $g = \infty$. 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. Note that at each time t the algorithm uses only information about jobs released by time t, so it is truly online.

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 at time t.
   (b) Run every unscheduled job j s.t. $[t, t + p_j) \subset T$; remove j from U.
   (c) If $t = u_j$ for some $j \in U$, then pick such a j with p_j maximal and set $T = T \cup [t, t + 2p_j)$ (activating the machine from time t to $t + 2p_j$). Let $P = P \cup \{j\}$.

(d) Run[3] 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 *Let T be the resulting active time and P the resulting set of primary jobs. Then $T = \bigcup_{j \in P} [s_j, s_j + 2p_j)$ and for every $j \in P$, $s_j = u_j$.*

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,

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

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.

[3] Step 2 (b) and step 2(d) are both necessary. Consider an interval job released at time 0 of length 2 and another at time 1 of length 4. Without step 2 (b) running at time 1, the machine will be turned on from time 5 to 9 unecessarily.

Now it follows that $p_{e^i_j} > 2p_{e^i_{j-1}}$ (for $j \geq 2$): suppose otherwise, then because we know e^i_j was available at step 2 (c) at $t = s(e^i_{j-1}) \geq s(e^i_1) \geq r_{e^i_j}$, job e^i_j must have been scheduled at t with e^i_{j-1} and cannot have been added to P. By contradiction, $p_{e^i_j} > 2p_{e^i_{j-1}}$ hence by induction $p_{e^i_{k'_i}} > 2^{k'_i - j} p_{e^i_j}$. Now (3) follows: $\sum_{j=1}^{k'_i} p_{e^i_j} \leq \sum_{j=1}^{k'_i} 2^{j-k'_i} p_{e^i_{k'_i}} < p_{e^i_{k'_i}} \sum_{j'=0}^{\infty} 2^{-j'} = 2p_{e^i_{k'_i}} \leq 2\mu(\mathcal{C}^i)$. 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.

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 [17] 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. Proposition 7 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.

Algorithm Compact

1. Find the minimum number of machine required to feasibly schedule the jobs by binary search ($0 \leq m_{opt} \leq n$), using a feasibility algorithm for the problem with mg identical single-processor machines. Then construct a schedule on S on these jobs and m_{opt} machines that minimizes the sum of completion times, $\sum C_j$. A $O(n^2)$ time algorithm for these tasks is known [14].
2. Let s_j^0 be the start time of job j in S, and let $s_j := s_j^0$, $K := \emptyset$ 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 $O(n \log n)$ time.

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

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, moving jobs rightward 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 first need to bucket jobs by similar lengths, 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. Below is our $(O(\log p_{max}/p_{min}), O(1))$-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.

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 analysis).
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.

Theorem 3. *The above algorithm generates a schedule feasible using* $(2\alpha + 1)OPT$ *busy time on* $\lceil \log_\alpha p_{max}/p_{min} \rceil (2\lceil \alpha \rceil m_{opt} + 8)$ *machines.*

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. In the previous section we could use that the $(r_j, d_j, p = p_j = 1)$ problem was exactly solvable offline; now, we must instead rely upon the online e-competitive online algorithm of [2] for this task. We also must use our $g = \infty$ algorithm in order to schedule the jobs in U'' online with bounded performance.

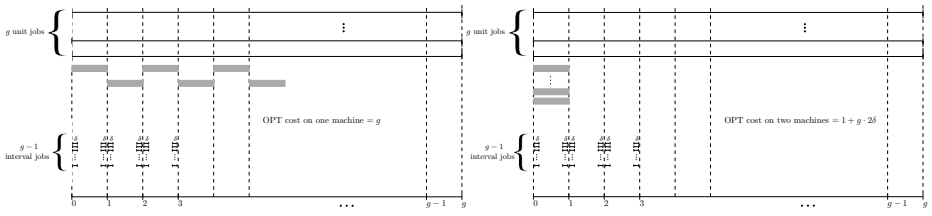
Theorem 4. *The online algorithm with lookahead* $3p_{max}$ *generates a schedule requiring at most* $\lceil \log_\alpha p_{max}/p_{min} \rceil (16 + 4\lceil e\lceil \alpha \rceil m_{opt} \rceil)$ *machines and* $(20 + 42\alpha)$ *busy time.*

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. *Algorithmica*, 70(3):368–405, November 2014.
5. Jessica Chang, Samir Khuller, and Koyel Mukherjee. Lp rounding and combinatorial algorithms for minimizing active and busy time. In *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. Xiaolin Fang, Hong Gao, Jianzhong Li, and Yingshu Li. Application-aware data collection in wireless sensor networks. In *Proceedings of INFOCOM*, 2013.
8. 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.
9. 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.
10. 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.
11. Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *FSTTCS*, pages 169 – 180, 2010.
12. Frederic Koehler and Samir Khuller. Optimal batch schedules for parallel machines. In *WADS*, pages 475–486, 2013.
13. Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *FSTTCS*, pages 152–163, 2005.
14. 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.
15. 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.
16. Runtian Ren and Xueyan Tang. Online flexible job scheduling for minimum span. In *SPAA*, 2017. To Appear.
17. 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.
18. Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *SODA*, pages 830 – 831, 2003.